BSDPort hotspot jdk7 changes from linux, os

Here are the hotspot os changes linux vs bsd:

```
diff -rus src/os/linux/vm/jsig.c src/os/bsd/vm/jsig.c
--- src/os/linux/vm/jsig.c        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/jsig.c        2011-07-26 20:21:21.000000000 -0600
@@ -35,10 +35,7 @@
 #include <pthread.h>
 #include <stdio.h>
 #include <stdlib.h>
-
-#define bool int
-#define true 1
-#define false 0
+#include <stdbool.h>

 #define MAXSIGNUM 32
 #define MASK(sig) ((unsigned int)1 << sig)
@@ -143,7 +140,8 @@
 }

 sa_handler_t sigset(int sig, sa_handler_t disp) {
-  return set_signal(sig, disp, true);
+  printf("sigset() is not supported by BSD");
+  exit(0);
  }

 static int call_os_sigaction(int sig, const struct sigaction  *act,
--- src/os/linux/vm/attachListener_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/attachListener_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -39,7 +39,7 @@
 #define UNIX_PATH_MAX   sizeof(((struct sockaddr_un *)0)->sun_path)
 #endif

-// The attach mechanism on Linux uses a UNIX domain socket. An attach listener
+// The attach mechanism on Bsd uses a UNIX domain socket. An attach listener
 // thread is created at startup or is created on-demand via a signal from
 // the client tool. The attach listener creates a socket and binds it to a file
 // in the filesystem. The attach listener then acts as a simple (single-
@@ -56,9 +56,9 @@
 //    of the client matches this process.

 // forward reference
-class LinuxAttachOperation;
+class BsdAttachOperation;

-class LinuxAttachListener: AllStatic {
+class BsdAttachListener: AllStatic {
  private:
   // the path to which we bind the UNIX domain socket
   static char _path[UNIX_PATH_MAX];
@@ -80,7 +80,7 @@
   static void set_listener(int s)               { _listener = s; }

   // reads a request from the given connected socket
-  static LinuxAttachOperation* read_request(int s);
+  static BsdAttachOperation* read_request(int s);

  public:
   enum {
@@ -100,10 +100,10 @@
   // write the given buffer to a socket
   static int write_fully(int s, char* buf, int len);

-  static LinuxAttachOperation* dequeue();
+  static BsdAttachOperation* dequeue();
 };
```

```
-class LinuxAttachOperation: public AttachOperation {
+class BsdAttachOperation: public AttachOperation {
  private:
   // the connection to the client
   int _socket;
@@ -114,15 +114,15 @@
   void set_socket(int s)                                { _socket = s; }
   int socket() const                                    { return _socket; }

-  LinuxAttachOperation(char* name) : AttachOperation(name) {
+  BsdAttachOperation(char* name) : AttachOperation(name) {
     set_socket(-1);
   }
 };

 // statics
-char LinuxAttachListener::_path[UNIX_PATH_MAX];
-bool LinuxAttachListener::_has_path;
-int LinuxAttachListener::_listener = -1;
+char BsdAttachListener::_path[UNIX_PATH_MAX];
+bool BsdAttachListener::_has_path;
+int BsdAttachListener::_listener = -1;

 // Supporting class to help split a buffer into individual components
 class ArgumentIterator : public StackObj {
@@ -156,12 +156,12 @@
     static int cleanup_done;
     if (!cleanup_done) {
       cleanup_done = 1;
-      int s = LinuxAttachListener::listener();
+      int s = BsdAttachListener::listener();
       if (s != -1) {
         ::close(s);
       }
-      if (LinuxAttachListener::has_path()) {
-        ::unlink(LinuxAttachListener::path());
+      if (BsdAttachListener::has_path()) {
+        ::unlink(BsdAttachListener::path());
       }
     }
   }
@@ -169,7 +169,7 @@

 // Initialization - create a listener socket and bind it to a file

-int LinuxAttachListener::init() {
+int BsdAttachListener::init() {
   char path[UNIX_PATH_MAX];          // socket file
   char initial_path[UNIX_PATH_MAX];  // socket file during setup
   int listener;                      // listener socket (file descriptor)
@@ -228,7 +228,7 @@
 // after the peer credentials have been checked and in the worst case it just
 // means that the attach listener thread is blocked.
 //
-LinuxAttachOperation* LinuxAttachListener::read_request(int s) {
+BsdAttachOperation* BsdAttachListener::read_request(int s) {
   char ver_str[8];
   sprintf(ver_str, "%d", ATTACH_PROTOCOL_VER);

@@ -298,7 +298,7 @@
     return NULL;
   }

-  LinuxAttachOperation* op = new LinuxAttachOperation(name);
+  BsdAttachOperation* op = new BsdAttachOperation(name);

   for (int i=0; i<AttachOperation::arg_count_max; i++) {
     char* arg = args.next();
@@ -320,10 +320,10 @@
```

```
 // Dequeue an operation
 //
-// In the Linux implementation there is only a single operation and clients
+// In the Bsd implementation there is only a single operation and clients
 // cannot queue commands (except at the socket level).
 //
-LinuxAttachOperation* LinuxAttachListener::dequeue() {
+BsdAttachOperation* BsdAttachListener::dequeue() {
   for (;;) {
     int s;

@@ -337,6 +337,15 @@

     // get the credentials of the peer and check the effective uid/guid
     // - check with jeff on this.
+#ifdef _ALLBSD_SOURCE
+    uid_t puid;
+    gid_t pgid;
+    if (::getpeereid(s, &puid, &pgid) != 0) {
+      int res;
+      RESTARTABLE(::close(s), res);
+      continue;
+    }
+#else
     struct ucred cred_info;
     socklen_t optlen = sizeof(cred_info);
     if (::getsockopt(s, SOL_SOCKET, SO_PEERCRED, (void*)&cred_info, &optlen) == -1) {
@@ -344,17 +353,20 @@
       RESTARTABLE(::close(s), res);
       continue;
     }
+    uid_t puid = cred_info.uid;
+    gid_t pgid = cred_info.gid;
+#endif
     uid_t euid = geteuid();
     gid_t egid = getegid();

-    if (cred_info.uid != euid || cred_info.gid != egid) {
+    if (puid != euid || pgid != egid) {
       int res;
       RESTARTABLE(::close(s), res);
       continue;
     }

     // peer credential look okay so we read the request
-    LinuxAttachOperation* op = read_request(s);
+    BsdAttachOperation* op = read_request(s);
     if (op == NULL) {
       int res;
       RESTARTABLE(::close(s), res);
@@ -366,7 +378,7 @@
 }

 // write the given buffer to the socket
-int LinuxAttachListener::write_fully(int s, char* buf, int len) {
+int BsdAttachListener::write_fully(int s, char* buf, int len) {
   do {
     int n = ::write(s, buf, len);
     if (n == -1) {
@@ -388,7 +400,7 @@
 // if there are operations that involves a very big reply then it the
 // socket could be made non-blocking and a timeout could be used.

-void LinuxAttachOperation::complete(jint result, bufferedStream* st) {
+void BsdAttachOperation::complete(jint result, bufferedStream* st) {
   JavaThread* thread = JavaThread::current();
   ThreadBlockInVM tbivm(thread);

@@ -399,11 +411,11 @@
   // write operation result
   char msg[32];
```

```
   sprintf(msg, "%d\n", result);
-  int rc = LinuxAttachListener::write_fully(this->socket(), msg, strlen(msg));
+  int rc = BsdAttachListener::write_fully(this->socket(), msg, strlen(msg));

   // write any result data
   if (rc == 0) {
-    LinuxAttachListener::write_fully(this->socket(), (char*) st->base(), st->size());
+    BsdAttachListener::write_fully(this->socket(), (char*) st->base(), st->size());
     ::shutdown(this->socket(), 2);
   }

@@ -427,7 +439,7 @@
   // cleared by handle_special_suspend_equivalent_condition() or
   // java_suspend_self() via check_and_wait_while_suspended()

-  AttachOperation* op = LinuxAttachListener::dequeue();
+  AttachOperation* op = BsdAttachListener::dequeue();

   // were we externally suspended while we were waiting?
   thread->check_and_wait_while_suspended();
@@ -443,7 +455,7 @@
   // cleared by handle_special_suspend_equivalent_condition() or
   // java_suspend_self() via check_and_wait_while_suspended()

-  int ret_code = LinuxAttachListener::init();
+  int ret_code = BsdAttachListener::init();

   // were we externally suspended while we were waiting?
   thread->check_and_wait_while_suspended();
@@ -467,16 +479,13 @@
   if (init_at_startup() || is_initialized()) {
     return false;               // initialized at startup or already initialized
   }
-  char fn[PATH_MAX+1];
-  sprintf(fn, ".attach_pid%d", os::current_process_id());
+  char path[PATH_MAX + 1];
   int ret;
-  struct stat64 st;
-  RESTARTABLE(::stat64(fn, &st), ret);
-  if (ret == -1) {
-    snprintf(fn, sizeof(fn), "%s/.attach_pid%d",
-             os::get_temp_directory(), os::current_process_id());
-    RESTARTABLE(::stat64(fn, &st), ret);
-  }
+  struct stat st;
+
+  snprintf(path, PATH_MAX + 1, "%s/.attach_pid%d",
+           os::get_temp_directory(), os::current_process_id());
+  RESTARTABLE(::stat(path, &st), ret);
   if (ret == 0) {
     // simple check to avoid starting the attach mechanism when
     // a bogus user creates the file
--- src/os/linux/vm/c1_globals_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/c1_globals_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,8 +22,8 @@
  *
  */

-#ifndef OS_LINUX_VM_C1_GLOBALS_LINUX_HPP
-#define OS_LINUX_VM_C1_GLOBALS_LINUX_HPP
+#ifndef OS_BSD_VM_C1_GLOBALS_BSD_HPP
+#define OS_BSD_VM_C1_GLOBALS_BSD_HPP

 #include "utilities/globalDefinitions.hpp"
 #include "utilities/macros.hpp"
@@ -33,4 +33,4 @@
 // client compiler. (see c1_globals.hpp)
 //

-#endif // OS_LINUX_VM_C1_GLOBALS_LINUX_HPP
+#endif // OS_BSD_VM_C1_GLOBALS_BSD_HPP
```

```
--- src/os/linux/vm/c2_globals_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/c2_globals_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,8 +22,8 @@
   *
   */

-#ifndef OS_LINUX_VM_C2_GLOBALS_LINUX_HPP
-#define OS_LINUX_VM_C2_GLOBALS_LINUX_HPP
+#ifndef OS_BSD_VM_C2_GLOBALS_BSD_HPP
+#define OS_BSD_VM_C2_GLOBALS_BSD_HPP

 #include "utilities/globalDefinitions.hpp"
 #include "utilities/macros.hpp"
@@ -33,4 +33,4 @@
 // server compiler. (see c2_globals.hpp)
 //

-#endif // OS_LINUX_VM_C2_GLOBALS_LINUX_HPP
+#endif // OS_BSD_VM_C2_GLOBALS_BSD_HPP
Files src/os/linux/vm/chaitin_linux.cpp and src/os/bsd/vm/chaitin_bsd.cpp are identical
Files src/os/linux/vm/decoder_linux.cpp and src/os/bsd/vm/decoder_bsd.cpp are identical
Files src/os/linux/vm/dtraceJSDT_linux.cpp and src/os/bsd/vm/dtraceJSDT_bsd.cpp are identical
--- src/os/linux/vm/globals_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/globals_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,19 +22,19 @@
   *
   */

-#ifndef OS_LINUX_VM_GLOBALS_LINUX_HPP
-#define OS_LINUX_VM_GLOBALS_LINUX_HPP
+#ifndef OS_BSD_VM_GLOBALS_BSD_HPP
+#define OS_BSD_VM_GLOBALS_BSD_HPP

 //
-// Defines Linux specific flags. They are not available on other platforms.
+// Defines Bsd specific flags. They are not available on other platforms.
 //
 #define RUNTIME_OS_FLAGS(develop, develop_pd, product, product_pd, diagnostic, notproduct) \
   product(bool, UseOprofile, false,                                         \
         "enable support for Oprofile profiler")                             \
                                                                             \
-  product(bool, UseLinuxPosixThreadCPUClocks, true,                         \
-          "enable fast Linux Posix clocks where available")                 \
-  /*  NB: The default value of UseLinuxPosixThreadCPUClocks may be          \
+  product(bool, UseBsdPosixThreadCPUClocks, true,                       \
+          "enable fast Bsd Posix clocks where available")               \
+  /*  NB: The default value of UseBsdPosixThreadCPUClocks may be          \
     overridden in Arguments::parse_each_vm_init_arg.  */                    \
                                                                             \
   product(bool, UseHugeTLBFS, false,                                        \
@@ -44,12 +44,12 @@
           "Use SYSV shared memory for large pages")

 //
-// Defines Linux-specific default values. The flags are available on all
+// Defines Bsd-specific default values. The flags are available on all
 // platforms, but they may have different default values on other platforms.
 //
-define_pd_global(bool, UseLargePages, true);
+define_pd_global(bool, UseLargePages, false);
 define_pd_global(bool, UseLargePagesIndividualAllocation, false);
 define_pd_global(bool, UseOSErrorReporting, false);
 define_pd_global(bool, UseThreadPriorities, true) ;

-#endif // OS_LINUX_VM_GLOBALS_LINUX_HPP
+#endif // OS_BSD_VM_GLOBALS_BSD_HPP
--- src/os/linux/vm/interfaceSupport_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/interfaceSupport_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,8 +22,8 @@
   *
   */
```

```
-#ifndef OS_LINUX_VM_INTERFACESUPPORT_LINUX_HPP
-#define OS_LINUX_VM_INTERFACESUPPORT_LINUX_HPP
+#ifndef OS_BSD_VM_INTERFACESUPPORT_BSD_HPP
+#define OS_BSD_VM_INTERFACESUPPORT_BSD_HPP

 // Contains inlined functions for class InterfaceSupport

@@ -31,4 +31,4 @@
   os::write_memory_serialize_page(thread);
 }

-#endif // OS_LINUX_VM_INTERFACESUPPORT_LINUX_HPP
+#endif // OS_BSD_VM_INTERFACESUPPORT_BSD_HPP
--- src/os/linux/vm/jvm_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/jvm_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -73,7 +73,7 @@
     case SHUTDOWN2_SIGNAL:
     case SHUTDOWN3_SIGNAL:
       if (ReduceSignalUsage) return (void*)-1;
-      if (os::Linux::is_sig_ignored(sig)) return (void*)1;
+      if (os::Bsd::is_sig_ignored(sig)) return (void*)1;
   }

   void* oldHandler = os::signal(sig, newHandler);
@@ -97,7 +97,7 @@
     }
   }
   else if ((sig == SHUTDOWN1_SIGNAL || sig == SHUTDOWN2_SIGNAL ||
-            sig == SHUTDOWN3_SIGNAL) && os::Linux::is_sig_ignored(sig)) {
+            sig == SHUTDOWN3_SIGNAL) && os::Bsd::is_sig_ignored(sig)) {
     // do not allow SHUTDOWN1_SIGNAL to be raised when SHUTDOWN1_SIGNAL
     // is ignored, since no handler for them is actually registered in JVM
     // or via JVM_RegisterSignal.
@@ -110,14 +110,14 @@
 JVM_END

 /*
-  All the defined signal names for Linux.
+  All the defined signal names for Bsd.

   NOTE that not all of these names are accepted by our Java implementation

   Via an existing claim by the VM, sigaction restrictions, or
   the "rules of Unix" some of these names will be rejected at runtime.
   For example the VM sets up to handle USR1, sigaction returns EINVAL for
-  STOP, and Linux simply doesn't allow catching of KILL.
+  STOP, and Bsd simply doesn't allow catching of KILL.

   Here are the names currently accepted by a user of sun.misc.Signal with
   1.4.1 (ignoring potential interaction with use of chaining, etc):
@@ -141,38 +141,31 @@
   "ILL",        SIGILL,         /* Illegal instruction (ANSI).  */
   "TRAP",       SIGTRAP,        /* Trace trap (POSIX).  */
   "ABRT",       SIGABRT,        /* Abort (ANSI).  */
-  "IOT",        SIGIOT,         /* IOT trap (4.2 BSD).  */
-  "BUS",        SIGBUS,         /* BUS error (4.2 BSD).  */
+  "EMT",        SIGEMT,                 /* EMT trap  */
   "FPE",        SIGFPE,         /* Floating-point exception (ANSI).  */
   "KILL",       SIGKILL,        /* Kill, unblockable (POSIX).  */
-  "USR1",       SIGUSR1,        /* User-defined signal 1 (POSIX).  */
+  "BUS",        SIGBUS,                 /* BUS error (4.2 BSD).  */
   "SEGV",       SIGSEGV,        /* Segmentation violation (ANSI).  */
-  "USR2",       SIGUSR2,        /* User-defined signal 2 (POSIX).  */
+  "SYS",        SIGSYS,                 /* Bad system call. Only on some Bsden! */
   "PIPE",       SIGPIPE,        /* Broken pipe (POSIX).  */
   "ALRM",       SIGALRM,        /* Alarm clock (POSIX).  */
   "TERM",       SIGTERM,        /* Termination (ANSI).  */
-#ifdef SIGSTKFLT
-  "STKFLT",     SIGSTKFLT,      /* Stack fault.  */
-#endif
```

```
-  "CLD",        SIGCLD,         /* Same as SIGCHLD (System V).  */
-  "CHLD",       SIGCHLD,        /* Child status has changed (POSIX).  */
-  "CONT",       SIGCONT,        /* Continue (POSIX).  */
+  "URG",        SIGURG,               /* Urgent condition on socket (4.2 BSD).  */
   "STOP",       SIGSTOP,        /* Stop, unblockable (POSIX).  */
   "TSTP",       SIGTSTP,        /* Keyboard stop (POSIX).  */
+  "CONT",        SIGCONT,        /* Continue (POSIX).  */
+  "CHLD",        SIGCHLD,        /* Child status has changed (POSIX).  */
   "TTIN",       SIGTTIN,        /* Background read from tty (POSIX).  */
   "TTOU",       SIGTTOU,        /* Background write to tty (POSIX).  */
-  "URG",        SIGURG,         /* Urgent condition on socket (4.2 BSD).  */
+  "IO",               SIGIO,               /* I/O now possible (4.2 BSD).  */
   "XCPU",       SIGXCPU,        /* CPU limit exceeded (4.2 BSD).  */
   "XFSZ",       SIGXFSZ,        /* File size limit exceeded (4.2 BSD).  */
   "VTALRM",     SIGVTALRM,      /* Virtual alarm clock (4.2 BSD).  */
   "PROF",       SIGPROF,        /* Profiling alarm clock (4.2 BSD).  */
   "WINCH",      SIGWINCH,       /* Window size change (4.3 BSD, Sun).  */
-  "POLL",       SIGPOLL,        /* Pollable event occurred (System V).  */
-  "IO",         SIGIO,          /* I/O now possible (4.2 BSD).  */
-  "PWR",        SIGPWR,         /* Power failure restart (System V).  */
-#ifdef SIGSYS
-  "SYS",        SIGSYS          /* Bad system call. Only on some Linuxen! */
-#endif
+  "INFO",        SIGINFO,        /* Information request.  */
+  "USR1",        SIGUSR1,        /* User-defined signal 1 (POSIX).  */
+  "USR2",        SIGUSR2               /* User-defined signal 2 (POSIX).  */
   };

 JVM_ENTRY_NO_ENV(jint, JVM_FindSignal(const char *name))
--- src/os/linux/vm/jvm_linux.h        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/jvm_bsd.h        2011-07-26 20:21:21.000000000 -0600
@@ -22,8 +22,8 @@
  *
  */

-#ifndef OS_LINUX_VM_JVM_LINUX_H
-#define OS_LINUX_VM_JVM_LINUX_H
+#ifndef OS_BSD_VM_JVM_BSD_H
+#define OS_BSD_VM_JVM_BSD_H

 /*
 // HotSpot integration note:
@@ -54,11 +54,15 @@
 #define AGENT_ONATTACH_SYMBOLS  {"Agent_OnAttach"}

 #define JNI_LIB_PREFIX "lib"
+#ifdef __APPLE__
+#define JNI_LIB_SUFFIX ".dylib"
+#else
 #define JNI_LIB_SUFFIX ".so"
+#endif

-// Hack: MAXPATHLEN is 4095 on some Linux and 4096 on others. This may
+// Hack: MAXPATHLEN is 4095 on some Bsd and 4096 on others. This may
 //       cause problems if JVM and the rest of JDK are built on different
-//       Linux releases. Here we define JVM_MAXPATHLEN to be MAXPATHLEN + 1,
+//       Bsd releases. Here we define JVM_MAXPATHLEN to be MAXPATHLEN + 1,
 //       so buffers declared in VM are always >= 4096.
 #define JVM_MAXPATHLEN MAXPATHLEN + 1

@@ -93,10 +97,24 @@
 #define SHUTDOWN2_SIGNAL SIGINT
 #define SHUTDOWN3_SIGNAL SIGTERM

+#ifndef SIGRTMIN
+#ifdef __OpenBSD__
+#define SIGRTMIN        1
+#else
+#define SIGRTMIN        33
+#endif
+#endif
```

```
+#ifndef SIGRTMAX
+#ifdef __OpenBSD__
+#define SIGRTMAX        31
+#else
+#define SIGRTMAX        63
+#endif
+#endif
 #endif /* JVM_MD_H */

 // Reconciliation History
 // jvm_solaris.h        1.6 99/06/22 16:38:47
 // End

-#endif // OS_LINUX_VM_JVM_LINUX_H
+#endif // OS_BSD_VM_JVM_BSD_H
--- src/os/linux/vm/mutex_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/mutex_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -23,10 +23,10 @@
   */

 #include "precompiled.hpp"
-#include "mutex_linux.inline.hpp"
+#include "mutex_bsd.inline.hpp"
 #include "runtime/interfaceSupport.hpp"
 #include "runtime/mutex.hpp"
-#include "thread_linux.inline.hpp"
+#include "thread_bsd.inline.hpp"
 #include "utilities/events.hpp"

 // put OS-includes here
--- src/os/linux/vm/mutex_linux.inline.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/mutex_bsd.inline.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,16 +22,16 @@
   *
   */

-#ifndef OS_LINUX_VM_MUTEX_LINUX_INLINE_HPP
-#define OS_LINUX_VM_MUTEX_LINUX_INLINE_HPP
+#ifndef OS_BSD_VM_MUTEX_BSD_INLINE_HPP
+#define OS_BSD_VM_MUTEX_BSD_INLINE_HPP

-#include "os_linux.inline.hpp"
+#include "os_bsd.inline.hpp"
 #include "runtime/interfaceSupport.hpp"
-#include "thread_linux.inline.hpp"
+#include "thread_bsd.inline.hpp"


 // Reconciliation History
 // mutex_solaris.inline.hpp     1.5 99/06/22 16:38:49
 // End

-#endif // OS_LINUX_VM_MUTEX_LINUX_INLINE_HPP
+#endif // OS_BSD_VM_MUTEX_BSD_INLINE_HPP
--- src/os/linux/vm/osThread_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/osThread_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -49,8 +49,8 @@

 void OSThread::pd_initialize() {
   assert(this != NULL, "check");
-  _thread_id        = 0;
-  _pthread_id       = 0;
+  _thread_id        = NULL;
+  _pthread_id       = NULL;
   _siginfo = NULL;
   _ucontext = NULL;
   _expanding_stack = 0;
--- src/os/linux/vm/osThread_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/osThread_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,8 +22,8 @@
   *
```

```
  */

-#ifndef OS_LINUX_VM_OSTHREAD_LINUX_HPP
-#define OS_LINUX_VM_OSTHREAD_LINUX_HPP
+#ifndef OS_BSD_VM_OSTHREAD_BSD_HPP
+#define OS_BSD_VM_OSTHREAD_BSD_HPP

  private:
   int _thread_type;
@@ -39,14 +39,21 @@

  private:

+#ifdef _ALLBSD_SOURCE
+  // _thread_id and _pthread_id are the same on BSD
+  // keep both to minimize code divergence in os_bsd.cpp
+  pthread_t _thread_id;
+  pthread_t _pthread_id;
+#else
   // _thread_id is kernel thread id (similar to LWP id on Solaris). Each
-  // thread has a unique thread_id (LinuxThreads or NPTL). It can be used
+  // thread has a unique thread_id (BsdThreads or NPTL). It can be used
   // to access /proc.
   pid_t     _thread_id;

   // _pthread_id is the pthread id, which is used by library calls
   // (e.g. pthread_kill).
   pthread_t _pthread_id;
+#endif

   sigset_t _caller_sigmask; // Caller's signal mask

@@ -56,12 +63,18 @@
   sigset_t  caller_sigmask() const       { return _caller_sigmask; }
   void    set_caller_sigmask(sigset_t sigmask)  { _caller_sigmask = sigmask; }

+#ifdef _ALLBSD_SOURCE
+  pthread_t thread_id() const {
+    return _thread_id;
+  }
+#else
   pid_t thread_id() const {
     return _thread_id;
   }
+#endif
 #ifndef PRODUCT
   // Used for debugging, return a unique integer for each thread.
-  int thread_identifier() const   { return _thread_id; }
+  intptr_t thread_identifier() const   { return (intptr_t)_pthread_id; }
 #endif
 #ifdef ASSERT
   // We expect no reposition failures so kill vm if we get one.
@@ -70,9 +83,15 @@
     return false;
   }
 #endif // ASSERT
+#ifdef _ALLBSD_SOURCE
+  void set_thread_id(pthread_t id) {
+    _thread_id = id;
+  }
+#else
   void set_thread_id(pid_t id) {
     _thread_id = id;
   }
+#endif
   pthread_t pthread_id() const {
     return _pthread_id;
   }
@@ -85,10 +104,10 @@
   // ************************************************************
```

```
 public:
-  // flags that support signal based suspend/resume on Linux are in a
+  // flags that support signal based suspend/resume on Bsd are in a
    // separate class to avoid confusion with many flags in OSThread that
    // are used by VM level suspend/resume.
-  os::Linux::SuspendResume sr;
+  os::Bsd::SuspendResume sr;

    // _ucontext and _siginfo are used by SR_handler() to save thread context,
    // and they will later be used to walk the stack or reposition thread PC.
@@ -143,4 +162,4 @@
 // osThread_solaris.hpp 1.24 99/08/27 13:11:54
 // End

-#endif // OS_LINUX_VM_OSTHREAD_LINUX_HPP
+#endif // OS_BSD_VM_OSTHREAD_BSD_HPP
--- src/os/linux/vm/os_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/os_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -30,12 +30,12 @@
 #include "code/vtableStubs.hpp"
 #include "compiler/compileBroker.hpp"
 #include "interpreter/interpreter.hpp"
-#include "jvm_linux.h"
+#include "jvm_bsd.h"
 #include "memory/allocation.inline.hpp"
 #include "memory/filemap.hpp"
-#include "mutex_linux.inline.hpp"
+#include "mutex_bsd.inline.hpp"
 #include "oops/oop.inline.hpp"
-#include "os_share_linux.hpp"
+#include "os_share_bsd.hpp"
 #include "prims/jniFastGetField.hpp"
 #include "prims/jvm.h"
 #include "prims/jvm_misc.hpp"
@@ -56,7 +56,7 @@
 #include "runtime/timer.hpp"
 #include "services/attachListener.hpp"
 #include "services/runtimeService.hpp"
-#include "thread_linux.inline.hpp"
+#include "thread_bsd.inline.hpp"
 #include "utilities/decoder.hpp"
 #include "utilities/defaultStream.hpp"
 #include "utilities/events.hpp"
@@ -108,21 +108,42 @@
 # include <sys/utsname.h>
 # include <sys/socket.h>
 # include <sys/wait.h>
+# include <time.h>
 # include <pwd.h>
 # include <poll.h>
 # include <semaphore.h>
 # include <fcntl.h>
 # include <string.h>
+#ifdef _ALLBSD_SOURCE
+# include <sys/param.h>
+# include <sys/sysctl.h>
+#else
 # include <syscall.h>
 # include <sys/sysinfo.h>
 # include <gnu/libc-version.h>
+#endif
 # include <sys/ipc.h>
 # include <sys/shm.h>
+#ifndef __APPLE__
 # include <link.h>
+#endif
 # include <stdint.h>
 # include <inttypes.h>
 # include <sys/ioctl.h>

+#if defined(__FreeBSD__) || defined(__NetBSD__)
```

```
+# include <elf.h>
+#endif
+
+#ifdef __APPLE__
+#include <mach/mach.h> // semaphore_* API
+#include <mach-o/dyld.h>
+#endif
+
+#ifndef MAP_ANONYMOUS
+#define MAP_ANONYMOUS MAP_ANON
+#endif
+
 #define MAX_PATH    (2 * K)

 // for timer info max values which include all bits
@@ -132,21 +153,27 @@
 #define LARGEPAGES_BIT (1 << 6)
 ////////////////////////////////////////////////////////////////////////////
 // global variables
-julong os::Linux::_physical_memory = 0;
+julong os::Bsd::_physical_memory = 0;

-address   os::Linux::_initial_thread_stack_bottom = NULL;
-uintptr_t os::Linux::_initial_thread_stack_size   = 0;
+#ifndef _ALLBSD_SOURCE
+address   os::Bsd::_initial_thread_stack_bottom = NULL;
+uintptr_t os::Bsd::_initial_thread_stack_size   = 0;
+#endif

-int (*os::Linux::_clock_gettime)(clockid_t, struct timespec *) = NULL;
-int (*os::Linux::_pthread_getcpuclockid)(pthread_t, clockid_t *) = NULL;
-Mutex* os::Linux::_createThread_lock = NULL;
-pthread_t os::Linux::_main_thread;
-int os::Linux::_page_size = -1;
-bool os::Linux::_is_floating_stack = false;
-bool os::Linux::_is_NPTL = false;
-bool os::Linux::_supports_fast_thread_cpu_time = false;
-const char * os::Linux::_glibc_version = NULL;
-const char * os::Linux::_libpthread_version = NULL;
+int (*os::Bsd::_clock_gettime)(clockid_t, struct timespec *) = NULL;
+#ifndef _ALLBSD_SOURCE
+int (*os::Bsd::_pthread_getcpuclockid)(pthread_t, clockid_t *) = NULL;
+Mutex* os::Bsd::_createThread_lock = NULL;
+#endif
+pthread_t os::Bsd::_main_thread;
+int os::Bsd::_page_size = -1;
+#ifndef _ALLBSD_SOURCE
+bool os::Bsd::_is_floating_stack = false;
+bool os::Bsd::_is_NPTL = false;
+bool os::Bsd::_supports_fast_thread_cpu_time = false;
+const char * os::Bsd::_glibc_version = NULL;
+const char * os::Bsd::_libpthread_version = NULL;
+#endif

 static jlong initial_time_count=0;

@@ -164,8 +191,6 @@
 static int SR_signum = SIGUSR2;
 sigset_t SR_sigset;

-/* Used to protect dlsym() calls */
-static pthread_mutex_t dl_mutex;

 ////////////////////////////////////////////////////////////////////////////
 // utility functions
@@ -174,19 +199,24 @@
 static int SR_finalize();

 julong os::available_memory() {
-  return Linux::available_memory();
+  return Bsd::available_memory();
```

```
 }

-julong os::Linux::available_memory() {
+julong os::Bsd::available_memory() {
+#ifdef _ALLBSD_SOURCE
+  // XXXBSD: this is just a stopgap implementation
+  return physical_memory() >> 2;
+#else
   // values in struct sysinfo are "unsigned long"
   struct sysinfo si;
   sysinfo(&si);

   return (julong)si.freeram * si.mem_unit;
+#endif
 }

 julong os::physical_memory() {
-  return Linux::physical_memory();
+  return Bsd::physical_memory();
 }

 julong os::allocatable_physical_memory(julong size) {
@@ -230,6 +260,7 @@
 }


+#ifndef _ALLBSD_SOURCE
 #ifndef SYS_gettid
 // i386: 224, ia64: 1105, amd64: 186, sparc 143
 #ifdef __ia64__
@@ -244,6 +275,7 @@
 #error define gettid for the arch
 #endif
 #endif
+#endif

 // Cpu architecture string
 #if   defined(ZERO)
@@ -269,15 +301,16 @@
 #endif


+#ifndef _ALLBSD_SOURCE
 // pid_t gettid()
 //
 // Returns the kernel thread id of the currently running thread. Kernel
 // thread id is used to access /proc.
 //
-// (Note that getpid() on LinuxThreads returns kernel thread id too; but
+// (Note that getpid() on BsdThreads returns kernel thread id too; but
 // on NPTL, it returns the same pid for all threads, as required by POSIX.)
 //
-pid_t os::Linux::gettid() {
+pid_t os::Bsd::gettid() {
   int rslt = syscall(SYS_gettid);
   if (rslt == -1) {
     // old kernel, no NPTL support
@@ -287,19 +320,60 @@
   }
 }

-// Most versions of linux have a bug where the number of processors are
+// Most versions of bsd have a bug where the number of processors are
 // determined by looking at the /proc file system.  In a chroot environment,
 // the system call returns 1.  This causes the VM to act as if it is
 // a single processor and elide locking (see is_MP() call).
 static bool unsafe_chroot_detected = false;
 static const char *unstable_chroot_error = "/proc file system not found.\n"
                     "Java may be unstable running multithreaded in a chroot "
-                    "environment on Linux when /proc filesystem is not mounted.";
+                    "environment on Bsd when /proc filesystem is not mounted.";
```

```
+#endif
+
+#ifdef _ALLBSD_SOURCE
+void os::Bsd::initialize_system_info() {
+  int mib[2];
+  size_t len;
+  int cpu_val;
+  u_long mem_val;
+
+  /* get processors count via hw.ncpus sysctl */
+  mib[0] = CTL_HW;
+  mib[1] = HW_NCPU;
+  len = sizeof(cpu_val);
+  if (sysctl(mib, 2, &cpu_val, &len, NULL, 0) != -1 && cpu_val >= 1) {
+       set_processor_count(cpu_val);
+  }
+  else {
+       set_processor_count(1);   // fallback
+  }
+
+  /* get physical memory via hw.usermem sysctl (hw.usermem is used
+   * instead of hw.physmem because we need size of allocatable memory
+   */
+  mib[0] = CTL_HW;
+  mib[1] = HW_USERMEM;
+  len = sizeof(mem_val);
+  if (sysctl(mib, 2, &mem_val, &len, NULL, 0) != -1)
+       _physical_memory = mem_val;
+  else
+       _physical_memory = 256*1024*1024;       // fallback (XXXBSD?)

-void os::Linux::initialize_system_info() {
+#ifdef __OpenBSD__
+  {
+       // limit _physical_memory memory view on OpenBSD since
+       // datasize rlimit restricts us anyway.
+       struct rlimit limits;
+       getrlimit(RLIMIT_DATA, &limits);
+       _physical_memory = MIN2(_physical_memory, (julong)limits.rlim_cur);
+  }
+#endif
+}
+#else
+void os::Bsd::initialize_system_info() {
   set_processor_count(sysconf(_SC_NPROCESSORS_CONF));
   if (processor_count() == 1) {
-    pid_t pid = os::Linux::gettid();
+    pid_t pid = os::Bsd::gettid();
     char fname[32];
     jio_snprintf(fname, sizeof(fname), "/proc/%d", pid);
     FILE *fp = fopen(fname, "r");
@@ -310,8 +384,9 @@
     }
   }
   _physical_memory = (julong)sysconf(_SC_PHYS_PAGES) * (julong)sysconf(_SC_PAGESIZE);
-  assert(processor_count() > 0, "linux error");
+  assert(processor_count() > 0, "bsd error");
 }
+#endif

 void os::init_system_properties_values() {
 //  char arch[12];
@@ -355,9 +430,7 @@
  *        ...
  *        7: The default directories, normally /lib and /usr/lib.
  */
-#if defined(AMD64) || defined(_LP64) && (defined(SPARC) || defined(PPC) || defined(S390))
-#define DEFAULT_LIBPATH "/usr/lib64:/lib64:/lib:/usr/lib"
-#else
+#ifndef DEFAULT_LIBPATH
 #define DEFAULT_LIBPATH "/lib:/usr/lib"
```

```
 #endif

@@ -411,7 +484,7 @@
      *
      * Note: Due to a legacy implementation, most of the library path
      * is set in the launcher.  This was to accomodate linking restrictions
-     * on legacy Linux implementations (which are no longer supported).
+     * on legacy Bsd implementations (which are no longer supported).
      * Eventually, all the library path setting will be done here.
      *
      * However, to prevent the proliferation of improperly built native
@@ -436,7 +509,11 @@
          * should always exist (until the legacy problem cited above is
          * addressed).
          */
+#ifdef __APPLE__
+        char *v = getenv("DYLD_LIBRARY_PATH");
+#else
         char *v = getenv("LD_LIBRARY_PATH");
+#endif
         if (v != NULL) {
             char *t = ld_library_path;
             /* That's +1 for the colon and +1 for the trailing '\0' */
@@ -496,7 +573,7 @@
 debug_only(static bool signal_sets_initialized = false);
 static sigset_t unblocked_sigs, vm_sigs, allowdebug_blocked_sigs;

-bool os::Linux::is_sig_ignored(int sig) {
+bool os::Bsd::is_sig_ignored(int sig) {
       struct sigaction oact;
       sigaction(sig, (struct sigaction*)NULL, &oact);
       void* ohlr = oact.sa_sigaction ? CAST_FROM_FN_PTR(void*,  oact.sa_sigaction)
@@ -507,7 +584,7 @@
           return false;
 }

-void os::Linux::signal_sets_init() {
+void os::Bsd::signal_sets_init() {
   // Should also have an assertion stating we are still single-threaded.
   assert(!signal_sets_initialized, "Already initialized");
   // Fill in signals that are necessarily unblocked for all threads in
@@ -532,15 +609,15 @@
   sigaddset(&unblocked_sigs, SR_signum);

   if (!ReduceSignalUsage) {
-   if (!os::Linux::is_sig_ignored(SHUTDOWN1_SIGNAL)) {
+   if (!os::Bsd::is_sig_ignored(SHUTDOWN1_SIGNAL)) {
       sigaddset(&unblocked_sigs, SHUTDOWN1_SIGNAL);
       sigaddset(&allowdebug_blocked_sigs, SHUTDOWN1_SIGNAL);
    }
-   if (!os::Linux::is_sig_ignored(SHUTDOWN2_SIGNAL)) {
+   if (!os::Bsd::is_sig_ignored(SHUTDOWN2_SIGNAL)) {
       sigaddset(&unblocked_sigs, SHUTDOWN2_SIGNAL);
       sigaddset(&allowdebug_blocked_sigs, SHUTDOWN2_SIGNAL);
    }
-   if (!os::Linux::is_sig_ignored(SHUTDOWN3_SIGNAL)) {
+   if (!os::Bsd::is_sig_ignored(SHUTDOWN3_SIGNAL)) {
       sigaddset(&unblocked_sigs, SHUTDOWN3_SIGNAL);
       sigaddset(&allowdebug_blocked_sigs, SHUTDOWN3_SIGNAL);
    }
@@ -555,25 +632,25 @@

 // These are signals that are unblocked while a thread is running Java.
 // (For some reason, they get blocked by default.)
-sigset_t* os::Linux::unblocked_signals() {
+sigset_t* os::Bsd::unblocked_signals() {
   assert(signal_sets_initialized, "Not initialized");
   return &unblocked_sigs;
 }

 // These are the signals that are blocked while a (non-VM) thread is
```

```
 // running Java. Only the VM thread handles these signals.
-sigset_t* os::Linux::vm_signals() {
+sigset_t* os::Bsd::vm_signals() {
   assert(signal_sets_initialized, "Not initialized");
   return &vm_sigs;
 }

 // These are signals that are blocked during cond_wait to allow debugger in
-sigset_t* os::Linux::allowdebug_blocked_signals() {
+sigset_t* os::Bsd::allowdebug_blocked_signals() {
   assert(signal_sets_initialized, "Not initialized");
   return &allowdebug_blocked_sigs;
 }

-void os::Linux::hotspot_sigmask(Thread* thread) {
+void os::Bsd::hotspot_sigmask(Thread* thread) {

   //Save caller's signal mask before setting VM signal mask
   sigset_t caller_sigmask;
@@ -582,7 +659,7 @@
   OSThread* osthread = thread->osthread();
   osthread->set_caller_sigmask(caller_sigmask);

-  pthread_sigmask(SIG_UNBLOCK, os::Linux::unblocked_signals(), NULL);
+  pthread_sigmask(SIG_UNBLOCK, os::Bsd::unblocked_signals(), NULL);

   if (!ReduceSignalUsage) {
     if (thread->is_VM_thread()) {
@@ -595,10 +672,11 @@
   }
 }

+#ifndef _ALLBSD_SOURCE
 ////////////////////////////////////////////////////////////////////////////////
 // detecting pthread library

-void os::Linux::libpthread_init() {
+void os::Bsd::libpthread_init() {
   // Save glibc and pthread version strings. Note that _CS_GNU_LIBC_VERSION
   // and _CS_GNU_LIBPTHREAD_VERSION are supported in glibc >= 2.3.2. Use a
   // generic name for earlier versions.
@@ -614,13 +692,13 @@
   if (n > 0) {
      char *str = (char *)malloc(n);
      confstr(_CS_GNU_LIBC_VERSION, str, n);
-     os::Linux::set_glibc_version(str);
+     os::Bsd::set_glibc_version(str);
   } else {
      // _CS_GNU_LIBC_VERSION is not supported, try gnu_get_libc_version()
      static char _gnu_libc_version[32];
      jio_snprintf(_gnu_libc_version, sizeof(_gnu_libc_version),
               "glibc %s %s", gnu_get_libc_version(), gnu_get_libc_release());
-     os::Linux::set_glibc_version(_gnu_libc_version);
+     os::Bsd::set_glibc_version(_gnu_libc_version);
   }

   n = confstr(_CS_GNU_LIBPTHREAD_VERSION, NULL, 0);
@@ -628,48 +706,48 @@
      char *str = (char *)malloc(n);
      confstr(_CS_GNU_LIBPTHREAD_VERSION, str, n);
      // Vanilla RH-9 (glibc 2.3.2) has a bug that confstr() always tells
-     // us "NPTL-0.29" even we are running with LinuxThreads. Check if this
-     // is the case. LinuxThreads has a hard limit on max number of threads.
+     // us "NPTL-0.29" even we are running with BsdThreads. Check if this
+     // is the case. BsdThreads has a hard limit on max number of threads.
      // So sysconf(_SC_THREAD_THREADS_MAX) will return a positive value.
      // On the other hand, NPTL does not have such a limit, sysconf()
      // will return -1 and errno is not changed. Check if it is really NPTL.
-     if (strcmp(os::Linux::glibc_version(), "glibc 2.3.2") == 0 &&
+     if (strcmp(os::Bsd::glibc_version(), "glibc 2.3.2") == 0 &&
          strstr(str, "NPTL") &&
```

```
          sysconf(_SC_THREAD_THREADS_MAX) > 0) {
        free(str);
-       os::Linux::set_libpthread_version("linuxthreads");
+       os::Bsd::set_libpthread_version("bsdthreads");
      } else {
-       os::Linux::set_libpthread_version(str);
+       os::Bsd::set_libpthread_version(str);
      }
   } else {
-    // glibc before 2.3.2 only has LinuxThreads.
-    os::Linux::set_libpthread_version("linuxthreads");
+    // glibc before 2.3.2 only has BsdThreads.
+    os::Bsd::set_libpthread_version("bsdthreads");
   }

   if (strstr(libpthread_version(), "NPTL")) {
-     os::Linux::set_is_NPTL();
+     os::Bsd::set_is_NPTL();
   } else {
-     os::Linux::set_is_LinuxThreads();
+     os::Bsd::set_is_BsdThreads();
   }

-  // LinuxThreads have two flavors: floating-stack mode, which allows variable
+  // BsdThreads have two flavors: floating-stack mode, which allows variable
   // stack size; and fixed-stack mode. NPTL is always floating-stack.
-  if (os::Linux::is_NPTL() || os::Linux::supports_variable_stack_size()) {
-     os::Linux::set_is_floating_stack();
+  if (os::Bsd::is_NPTL() || os::Bsd::supports_variable_stack_size()) {
+     os::Bsd::set_is_floating_stack();
   }
 }

 ////////////////////////////////////////////////////////////////////////////////
 // thread stack

-// Force Linux kernel to expand current thread stack. If "bottom" is close
+// Force Bsd kernel to expand current thread stack. If "bottom" is close
 // to the stack guard, caller should block all signals.
 //
 // MAP_GROWSDOWN:
 //   A special mmap() flag that is used to implement thread stacks. It tells
 //   kernel that the memory region should extend downwards when needed. This
-//   allows early versions of LinuxThreads to only mmap the first few pages
-//   when creating a new thread. Linux kernel will automatically expand thread
+//   allows early versions of BsdThreads to only mmap the first few pages
+//   when creating a new thread. Bsd kernel will automatically expand thread
 //   stack as needed (on page faults).
 //
 //   However, because the memory region of a MAP_GROWSDOWN stack can grow on
@@ -677,19 +755,19 @@
 //   region, it's hard to tell if the fault is due to a legitimate stack
 //   access or because of reading/writing non-exist memory (e.g. buffer
 //   overrun). As a rule, if the fault happens below current stack pointer,
-//   Linux kernel does not expand stack, instead a SIGSEGV is sent to the
-//   application (see Linux kernel fault.c).
+//   Bsd kernel does not expand stack, instead a SIGSEGV is sent to the
+//   application (see Bsd kernel fault.c).
 //
-//   This Linux feature can cause SIGSEGV when VM bangs thread stack for
+//   This Bsd feature can cause SIGSEGV when VM bangs thread stack for
 //   stack overflow detection.
 //
-//   Newer version of LinuxThreads (since glibc-2.2, or, RH-7.x) and NPTL do
+//   Newer version of BsdThreads (since glibc-2.2, or, RH-7.x) and NPTL do
 //   not use this flag. However, the stack of initial thread is not created
 //   by pthread, it is still MAP_GROWSDOWN. Also it's possible (though
 //   unlikely) that user code can create a thread with MAP_GROWSDOWN stack
 //   and then attach the thread to JVM.
 //
-// To get around the problem and allow stack banging on Linux, we need to
```

```
+// To get around the problem and allow stack banging on Bsd, we need to
 // manually expand thread stack after receiving the SIGSEGV.
 //
 // There are two ways to expand thread stack to address "bottom", we used
@@ -705,7 +783,7 @@
 // That will destroy the mmap() frame and cause VM to crash.
 //
 // The following code works by adjusting sp first, then accessing the "bottom"
-// page to force a page fault. Linux kernel will then automatically expand the
+// page to force a page fault. Bsd kernel will then automatically expand the
 // stack mapping.
 //
 // _expand_stack_to() assumes its frame size is less than page size, which
@@ -726,8 +804,8 @@

   // Adjust bottom to point to the largest address within the same page, it
   // gives us a one-page buffer if alloca() allocates slightly more memory.
-  bottom = (address)align_size_down((uintptr_t)bottom, os::Linux::page_size());
-  bottom += os::Linux::page_size() - 1;
+  bottom = (address)align_size_down((uintptr_t)bottom, os::Bsd::page_size());
+  bottom += os::Bsd::page_size() - 1;

   // sp might be slightly above current stack pointer; if that's the case, we
   // will alloca() a little more space than necessary, which is OK. Don't use
@@ -743,7 +821,7 @@
   }
 }

-bool os::Linux::manually_expand_stack(JavaThread * t, address addr) {
+bool os::Bsd::manually_expand_stack(JavaThread * t, address addr) {
   assert(t!=NULL, "just checking");
   assert(t->osthread()->expanding_stack(), "expand should be set");
   assert(t->stack_base() != NULL, "stack_base was not initialized");
@@ -758,6 +836,7 @@
   }
   return false;
 }
+#endif

 ////////////////////////////////////////////////////////////////////////////////
 // create new thread
@@ -766,12 +845,15 @@

 // check if it's safe to start a new thread
 static bool _thread_safety_check(Thread* thread) {
-  if (os::Linux::is_LinuxThreads() && !os::Linux::is_floating_stack()) {
-    // Fixed stack LinuxThreads (SuSE Linux/x86, and some versions of Redhat)
+#ifdef _ALLBSD_SOURCE
+    return true;
+#else
+  if (os::Bsd::is_BsdThreads() && !os::Bsd::is_floating_stack()) {
+    // Fixed stack BsdThreads (SuSE Bsd/x86, and some versions of Redhat)
     //   Heap is mmap'ed at lower end of memory space. Thread stacks are
     //   allocated (MAP_FIXED) from high address space. Every thread stack
     //   occupies a fixed size slot (usually 2Mbytes, but user can change
-    //   it to other values if they rebuild LinuxThreads).
+    //   it to other values if they rebuild BsdThreads).
     //
     // Problem with MAP_FIXED is that mmap() can still succeed even part of
     // the memory region has already been mmap'ed. That means if we have too
@@ -793,12 +875,13 @@
       return true;
     }
   } else {
-    // Floating stack LinuxThreads or NPTL:
-    //   Unlike fixed stack LinuxThreads, thread stacks are not MAP_FIXED. When
+    // Floating stack BsdThreads or NPTL:
+    //   Unlike fixed stack BsdThreads, thread stacks are not MAP_FIXED. When
     //   there's not enough space left, pthread_create() will fail. If we come
     //   here, that means enough space has been reserved for stack.
     return true;
```

```
   }
+#endif
 }

 // Thread start routine for all newly created threads
@@ -817,7 +900,7 @@
   OSThread* osthread = thread->osthread();
   Monitor* sync = osthread->startThread_lock();

-  // non floating stack LinuxThreads needs extra check, see above
+  // non floating stack BsdThreads needs extra check, see above
   if (!_thread_safety_check(thread)) {
     // notify parent thread
     MutexLockerEx ml(sync, Mutex::_no_safepoint_check_flag);
@@ -826,8 +909,12 @@
     return NULL;
   }

+#ifdef _ALLBSD_SOURCE
+  // thread_id is pthread_id on BSD
+  osthread->set_thread_id(::pthread_self());
+#else
   // thread_id is kernel thread id (similar to Solaris LWP id)
-  osthread->set_thread_id(os::Linux::gettid());
+  osthread->set_thread_id(os::Bsd::gettid());

   if (UseNUMA) {
     int lgrp_id = os::numa_get_group_id();
@@ -835,11 +922,12 @@
       thread->set_lgrp_id(lgrp_id);
     }
   }
+#endif
   // initialize signal mask for this thread
-  os::Linux::hotspot_sigmask(thread);
+  os::Bsd::hotspot_sigmask(thread);

   // initialize floating point control register
-  os::Linux::init_thread_fpu_state();
+  os::Bsd::init_thread_fpu_state();

   // handshaking with parent thread
   {
@@ -884,10 +972,10 @@
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

   // stack size
-  if (os::Linux::supports_variable_stack_size()) {
+  if (os::Bsd::supports_variable_stack_size()) {
     // calculate stack size if it's not specified by caller
     if (stack_size == 0) {
-      stack_size = os::Linux::default_stack_size(thr_type);
+      stack_size = os::Bsd::default_stack_size(thr_type);

       switch (thr_type) {
       case os::java_thread:
@@ -911,23 +999,28 @@
       }
     }

-    stack_size = MAX2(stack_size, os::Linux::min_stack_allowed);
+    stack_size = MAX2(stack_size, os::Bsd::min_stack_allowed);
     pthread_attr_setstacksize(&attr, stack_size);
   } else {
     // let pthread_create() pick the default value.
   }

+#ifndef _ALLBSD_SOURCE
   // glibc guard page
-  pthread_attr_setguardsize(&attr, os::Linux::default_guard_size(thr_type));
+  pthread_attr_setguardsize(&attr, os::Bsd::default_guard_size(thr_type));
```

```
+#endif

   ThreadState state;

   {
-    // Serialize thread creation if we are running with fixed stack LinuxThreads
-    bool lock = os::Linux::is_LinuxThreads() && !os::Linux::is_floating_stack();
+
+#ifndef _ALLBSD_SOURCE
+    // Serialize thread creation if we are running with fixed stack BsdThreads
+    bool lock = os::Bsd::is_BsdThreads() && !os::Bsd::is_floating_stack();
     if (lock) {
-      os::Linux::createThread_lock()->lock_without_safepoint_check();
+      os::Bsd::createThread_lock()->lock_without_safepoint_check();
     }
+#endif

     pthread_t tid;
     int ret = pthread_create(&tid, &attr, (void* (*)(void*)) java_start, thread);
@@ -941,7 +1034,9 @@
       // Need to clean up stuff we've allocated so far
       thread->set_osthread(NULL);
       delete osthread;
-      if (lock) os::Linux::createThread_lock()->unlock();
++#ifndef _ALLBSD_SOURCE
+      if (lock) os::Bsd::createThread_lock()->unlock();
++#endif
       return false;
     }

@@ -957,9 +1052,11 @@
       }
     }

+#ifndef _ALLBSD_SOURCE
     if (lock) {
-      os::Linux::createThread_lock()->unlock();
+      os::Bsd::createThread_lock()->unlock();
     }
+#endif
   }

   // Aborted due to thread limit being reached
@@ -980,7 +1077,7 @@

 // bootstrap the main thread
 bool os::create_main_thread(JavaThread* thread) {
-  assert(os::Linux::_main_thread == pthread_self(), "should be called inside main thread");
+  assert(os::Bsd::_main_thread == pthread_self(), "should be called inside main thread");
   return create_attached_thread(thread);
 }

@@ -997,17 +1094,22 @@
   }

   // Store pthread info into the OSThread
-  osthread->set_thread_id(os::Linux::gettid());
+#ifdef _ALLBSD_SOURCE
+  osthread->set_thread_id(::pthread_self());
+#else
+  osthread->set_thread_id(os::Bsd::gettid());
+#endif
   osthread->set_pthread_id(::pthread_self());

   // initialize floating point control register
-  os::Linux::init_thread_fpu_state();
+  os::Bsd::init_thread_fpu_state();

   // Initial thread state is RUNNABLE
   osthread->set_state(RUNNABLE);
```

```
   thread->set_osthread(osthread);

+#ifndef _ALLBSD_SOURCE
   if (UseNUMA) {
     int lgrp_id = os::numa_get_group_id();
     if (lgrp_id != -1) {
@@ -1015,14 +1117,14 @@
     }
   }

-  if (os::Linux::is_initial_thread()) {
+  if (os::Bsd::is_initial_thread()) {
     // If current thread is initial thread, its stack is mapped on demand,
     // see notes about MAP_GROWSDOWN. Here we try to force kernel to map
     // the entire stack region to avoid SEGV in stack banging.
     // It is also useful to get around the heap-stack-gap problem on SuSE
     // kernel (see 4821821 for details). We first expand stack to the top
     // of yellow zone, then enable stack yellow zone (order is significant,
-    // enabling yellow zone first will crash JVM on SuSE Linux), so there
+    // enabling yellow zone first will crash JVM on SuSE Bsd), so there
     // is no gap between the last two virtual memory regions.

     JavaThread *jt = (JavaThread *)thread;
@@ -1031,13 +1133,14 @@
     assert(jt->stack_available(addr) > 0, "stack guard should not be enabled");

     osthread->set_expanding_stack();
-    os::Linux::manually_expand_stack(jt, addr);
+    os::Bsd::manually_expand_stack(jt, addr);
     osthread->clear_expanding_stack();
   }
+#endif

   // initialize signal mask for this thread
   // and save the caller's signal mask
-  os::Linux::hotspot_sigmask(thread);
+  os::Bsd::hotspot_sigmask(thread);

   return true;
 }
@@ -1050,7 +1153,7 @@
   sync_with_child->notify();
 }

-// Free Linux resources related to the OSThread
+// Free Bsd resources related to the OSThread
 void os::free_thread(OSThread* osthread) {
   assert(osthread != NULL, "osthread not set");

@@ -1092,8 +1195,9 @@
 ////////////////////////////////////////////////////////////////////////////////
 // initial thread

+#ifndef _ALLBSD_SOURCE
 // Check if current thread is the initial thread, similar to Solaris thr_main.
-bool os::Linux::is_initial_thread(void) {
+bool os::Bsd::is_initial_thread(void) {
   char dummy;
   // If called before init complete, thread stack bottom will be null.
   // Can be called if fatal error occurs before initialization.
@@ -1132,9 +1236,9 @@
 }

 // Locate initial thread stack. This special handling of initial thread stack
-// is needed because pthread_getattr_np() on most (all?) Linux distros returns
+// is needed because pthread_getattr_np() on most (all?) Bsd distros returns
 // bogus value for initial thread.
-void os::Linux::capture_initial_stack(size_t max_size) {
+void os::Bsd::capture_initial_stack(size_t max_size) {
   // stack size is the easy part, get it from RLIMIT_STACK
   size_t stack_size;
```

```
   struct rlimit rlim;
@@ -1168,7 +1272,7 @@
   // variable is available since the very early days. However, because it is
   // a private interface, it could disappear in the future.
   //
-  // Linux kernel saves start_stack information in /proc/<pid>/stat. Similar
+  // Bsd kernel saves start_stack information in /proc/<pid>/stat. Similar
   // to __libc_stack_end, it is very close to stack top, but isn't the real
   // stack top. Note that /proc may not exist if VM is running as a chroot
   // program, so reading /proc/<pid>/stat could fail. Also the contents of
@@ -1286,7 +1390,7 @@
       }
     } else {
       // For some reason we can't open /proc/self/stat (for example, running on
-      // FreeBSD with a Linux emulator, or inside chroot), this should work for
+      // FreeBSD with a Bsd emulator, or inside chroot), this should work for
       // most cases, so don't abort:
       warning("Can't detect initial thread stack location - no /proc/self/stat");
       stack_start = (uintptr_t) &rlim;
@@ -1328,6 +1432,7 @@
   _initial_thread_stack_size = align_size_down(_initial_thread_stack_size, page_size());
   _initial_thread_stack_bottom = (address)stack_top - _initial_thread_stack_size;
 }
+#endif

 ////////////////////////////////////////////////////////////////////////////////
 // time support
@@ -1349,9 +1454,7 @@
   return (1000 * 1000);
 }

-// For now, we say that linux does not support vtime.  I have no idea
-// whether it can actually be made to (DLD, 9/13/05).
-
+// XXX: For now, code this as if BSD does not support vtime.
 bool os::supports_vtime() { return false; }
 bool os::enable_vtime()   { return false; }
 bool os::vtime_enabled()  { return false; }
@@ -1363,7 +1466,7 @@
   timeval time;
   int status = gettimeofday(&time, NULL);
-  assert(status != -1, "linux error");
+  assert(status != -1, "bsd error");
   return jlong(time.tv_sec) * 1000  +  jlong(time.tv_usec / 1000);
 }

@@ -1371,8 +1474,23 @@
 #define CLOCK_MONOTONIC (1)
 #endif

-void os::Linux::clock_init() {
-  // we do dlopen's in this particular order due to bug in linux
+#ifdef __APPLE__
+void os::Bsd::clock_init() {
+        // XXXDARWIN: Investigate replacement monotonic clock
+}
+#elif defined(_ALLBSD_SOURCE)
+void os::Bsd::clock_init() {
+  struct timespec res;
+  struct timespec tp;
+  if (::clock_getres(CLOCK_MONOTONIC, &res) == 0 &&
+      ::clock_gettime(CLOCK_MONOTONIC, &tp)  == 0) {
+    // yes, monotonic clock is supported
+    _clock_gettime = ::clock_gettime;
+  }
+}
+#else
+void os::Bsd::clock_init() {
+  // we do dlopen's in this particular order due to bug in bsd
   // dynamical loader (see 6348968) leading to crash on exit
```

```
   void* handle = dlopen("librt.so.1", RTLD_LAZY);
   if (handle == NULL) {
@@ -1406,7 +1524,9 @@
     }
   }
 }
+#endif

+#ifndef _ALLBSD_SOURCE
 #ifndef SYS_clock_getres

 #if defined(IA32) || defined(AMD64)
@@ -1421,8 +1541,8 @@
 #define sys_clock_getres(x,y)  ::syscall(SYS_clock_getres, x, y)
 #endif

-void os::Linux::fast_thread_clock_init() {
-  if (!UseLinuxPosixThreadCPUClocks) {
+void os::Bsd::fast_thread_clock_init() {
+  if (!UseBsdPosixThreadCPUClocks) {
     return;
   }
   clockid_t clockid;
@@ -1447,25 +1567,26 @@
     _pthread_getcpuclockid = pthread_getcpuclockid_func;
   }
 }
+#endif

 jlong os::javaTimeNanos() {
-  if (Linux::supports_monotonic_clock()) {
+  if (Bsd::supports_monotonic_clock()) {
     struct timespec tp;
-    int status = Linux::clock_gettime(CLOCK_MONOTONIC, &tp);
+    int status = Bsd::clock_gettime(CLOCK_MONOTONIC, &tp);
     assert(status == 0, "gettime error");
     jlong result = jlong(tp.tv_sec) * (1000 * 1000 * 1000) + jlong(tp.tv_nsec);
     return result;
   } else {
     timeval time;
     int status = gettimeofday(&time, NULL);
-    assert(status != -1, "linux error");
+    assert(status != -1, "bsd error");
     jlong usecs = jlong(time.tv_sec) * (1000 * 1000) + jlong(time.tv_usec);
     return 1000 * usecs;
   }
 }

 void os::javaTimeNanos_info(jvmtiTimerInfo *info_ptr) {
-  if (Linux::supports_monotonic_clock()) {
+  if (Bsd::supports_monotonic_clock()) {
     info_ptr->max_value = ALL_64_BITS;

     // CLOCK_MONOTONIC - amount of time since some arbitrary point in the past
@@ -1566,11 +1687,11 @@

 // Die immediately, no exit hook, no abort hook, no cleanup.
 void os::die() {
-  // _exit() on LinuxThreads only kills current thread
+  // _exit() on BsdThreads only kills current thread
   ::abort();
 }

-// unused on linux for now.
+// unused on bsd for now.
 void os::set_error_file(const char *logfile) {}


@@ -1594,10 +1715,10 @@
 intx os::current_thread_id() { return (intx)pthread_self(); }
 int os::current_process_id() {
```

```
-  // Under the old linux thread library, linux gives each thread
+  // Under the old bsd thread library, bsd gives each thread
   // its own process id. Because of this each thread will return
   // a different pid if this method were to return the result
-  // of getpid(2). Linux provides no api that returns the pid
+  // of getpid(2). Bsd provides no api that returns the pid
   // of the launcher thread for the vm. This implementation
   // returns a unique pid, the pid of the launcher thread
   // that starts the vm 'process'.
@@ -1608,14 +1729,21 @@

   // if you are looking for the result of a call to getpid() that
   // returns a unique pid for the calling thread, then look at the
-  // OSThread::thread_id() method in osThread_linux.hpp file
+  // OSThread::thread_id() method in osThread_bsd.hpp file

   return (int)(_initial_pid ? _initial_pid : getpid());
 }

 // DLL functions

-const char* os::dll_file_extension() { return ".so"; }
+#define JNI_LIB_PREFIX "lib"
+#ifdef __APPLE__
+#define JNI_LIB_SUFFIX ".dylib"
+#else
+#define JNI_LIB_SUFFIX ".so"
+#endif
+
+const char* os::dll_file_extension() { return JNI_LIB_SUFFIX; }

 // This must be hard coded because it's the system's temporary
 // directory not the java application's temp directory, ala java.io.tmpdir.
@@ -1635,13 +1763,13 @@
   const size_t pnamelen = pname ? strlen(pname) : 0;

   // Quietly truncate on buffer overflow.  Should be an error.
-  if (pnamelen + strlen(fname) + 10 > (size_t) buflen) {
+  if (pnamelen + strlen(fname) + strlen(JNI_LIB_PREFIX) + strlen(JNI_LIB_SUFFIX) + 2 > buflen) {
       *buffer = '\0';
       return;
   }

   if (pnamelen == 0) {
-    snprintf(buffer, buflen, "lib%s.so", fname);
+    snprintf(buffer, buflen, JNI_LIB_PREFIX "%s" JNI_LIB_SUFFIX, fname);
   } else if (strchr(pname, *os::path_separator()) != NULL) {
     int n;
     char** pelements = split_path(pname, &n);
@@ -1650,7 +1778,8 @@
       if (pelements[i] == NULL || strlen(pelements[i]) == 0) {
         continue; // skip the empty path values
       }
-      snprintf(buffer, buflen, "%s/lib%s.so", pelements[i], fname);
+      snprintf(buffer, buflen, "%s/" JNI_LIB_PREFIX "%s" JNI_LIB_SUFFIX,
+          pelements[i], fname);
       if (file_exists(buffer)) {
         break;
       }
@@ -1665,7 +1794,7 @@
       FREE_C_HEAP_ARRAY(char*, pelements);
     }
   } else {
-    snprintf(buffer, buflen, "%s/lib%s.so", pname, fname);
+    snprintf(buffer, buflen, "%s/" JNI_LIB_PREFIX "%s" JNI_LIB_SUFFIX, pname, fname);
   }
 }

@@ -1715,6 +1844,23 @@
   return false;
```

```
 }

+#ifdef _ALLBSD_SOURCE
+// ported from solaris version
+bool os::dll_address_to_library_name(address addr, char* buf,
+                                     int buflen, int* offset) {
+  Dl_info dlinfo;
+
+  if (dladdr((void*)addr, &dlinfo)){
+     if (buf) jio_snprintf(buf, buflen, "%s", dlinfo.dli_fname);
+     if (offset) *offset = addr - (address)dlinfo.dli_fbase;
+     return true;
+  } else {
+     if (buf) buf[0] = '\0';
+     if (offset) *offset = -1;
+     return false;
+  }
+}
+#else
 struct _address_to_library_name {
   address addr;          // input : memory address
   size_t  buflen;        //         size of fname
@@ -1789,11 +1935,27 @@
     return false;
   }
 }
+#endif

   // Loads .dll/.so and
   // in case of error it checks if .dll/.so was built for the
   // same architecture as Hotspot is running on
+
+#ifdef __APPLE__
+void * os::dll_load(const char *filename, char *ebuf, int ebuflen) {
+  void * result= ::dlopen(filename, RTLD_LAZY);
+  if (result != NULL) {
+    // Successful loading
+    return result;
+  }
+
+  // Read system error message into ebuf
+  ::strncpy(ebuf, ::dlerror(), ebuflen-1);
+  ebuf[ebuflen-1]='\0';

+  return NULL;
+}
+#else
 void * os::dll_load(const char *filename, char *ebuf, int ebuflen)
 {
   void * result= ::dlopen(filename, RTLD_LAZY);
@@ -1846,6 +2008,26 @@
   #define EM_486          6               /* Intel 80486 */
   #endif

+  #ifndef EM_MIPS_RS3_LE
+  #define EM_MIPS_RS3_LE  10              /* MIPS */
+  #endif
+
+  #ifndef EM_PPC64
+  #define EM_PPC64        21              /* PowerPC64 */
+  #endif
+
+  #ifndef EM_S390
+  #define EM_S390         22              /* IBM System/390 */
+  #endif
+
+  #ifndef EM_IA_64
+  #define EM_IA_64        50              /* HP/Intel IA-64 */
+  #endif
+
+  #ifndef EM_X86_64
```

```
+  #define EM_X86_64       62              /* AMD x86-64 */
+  #endif
+
   static const arch_t arch_array[]={
     {EM_386,         EM_386,     ELFCLASS32, ELFDATA2LSB, (char*)"IA 32"},
     {EM_486,         EM_386,     ELFCLASS32, ELFDATA2LSB, (char*)"IA 32"},
@@ -1949,17 +2131,11 @@

   return NULL;
 }
+#endif /* !__APPLE__ */

-/*
- * glibc-2.0 libdl is not MT safe.  If you are building with any glibc,
- * chances are you might want to run the generated bits against glibc-2.0
- * libdl.so, so always use locking for any version of glibc.
- */
+// XXX: Do we need a lock around this as per Linux?
 void* os::dll_lookup(void* handle, const char* name) {
-  pthread_mutex_lock(&dl_mutex);
-  void* res = dlsym(handle, name);
-  pthread_mutex_unlock(&dl_mutex);
-  return res;
+  return dlsym(handle, name);
 }


@@ -1982,15 +2158,60 @@

 void os::print_dll_info(outputStream *st) {
    st->print_cr("Dynamic libraries:");
-
+#ifdef _ALLBSD_SOURCE
+#ifdef RTLD_DI_LINKMAP
+    Dl_info dli;
+    void *handle;
+    Link_map *map;
+    Link_map *p;
+
+    if (!dladdr(CAST_FROM_FN_PTR(void *, os::print_dll_info), &dli)) {
+        st->print_cr("Error: Cannot print dynamic libraries.");
+        return;
+    }
+    handle = dlopen(dli.dli_fname, RTLD_LAZY);
+    if (handle == NULL) {
+        st->print_cr("Error: Cannot print dynamic libraries.");
+        return;
+    }
+    dlinfo(handle, RTLD_DI_LINKMAP, &map);
+    if (map == NULL) {
+        st->print_cr("Error: Cannot print dynamic libraries.");
+        return;
+    }
+
+    while (map->l_prev != NULL)
+        map = map->l_prev;
+
+    while (map != NULL) {
+        st->print_cr(PTR_FORMAT " \t%s", map->l_addr, map->l_name);
+        map = map->l_next;
+    }
+
+    dlclose(handle);
+#elif defined(__APPLE__)
+    uint32_t count;
+    uint32_t i;
+
+    count = _dyld_image_count();
+    for (i = 1; i < count; i++) {
+        const char *name = _dyld_get_image_name(i);
+        intptr_t slide = _dyld_get_image_vmaddr_slide(i);
```

```
+        st->print_cr(PTR_FORMAT " \t%s", slide, name);
+    }
+#else
+   st->print_cr("Error: Cannot print dynamic libraries.");
+#endif
+#else
    char fname[32];
-   pid_t pid = os::Linux::gettid();
+   pid_t pid = os::Bsd::gettid();

    jio_snprintf(fname, sizeof(fname), "/proc/%d/maps", pid);

    if (!_print_ascii_file(fname, st)) {
      st->print("Can not get library information for pid = %d\n", pid);
    }
+#endif
 }


@@ -1998,7 +2219,7 @@
   st->print("OS:");

   // Try to identify popular distros.
-  // Most Linux distributions have /etc/XXX-release file, which contains
+  // Most Bsd distributions have /etc/XXX-release file, which contains
   // the OS version string. Some have more than one /etc/XXX-release file
   // (e.g. Mandrake has both /etc/mandrake-release and /etc/redhat-release.),
   // so the order is important.
@@ -2006,12 +2227,12 @@
       !_print_ascii_file("/etc/sun-release", st) &&
       !_print_ascii_file("/etc/redhat-release", st) &&
       !_print_ascii_file("/etc/SuSE-release", st) &&
-      !_print_ascii_file("/etc/turbolinux-release", st) &&
+      !_print_ascii_file("/etc/turbobsd-release", st) &&
       !_print_ascii_file("/etc/gentoo-release", st) &&
       !_print_ascii_file("/etc/debian_version", st) &&
       !_print_ascii_file("/etc/ltib-release", st) &&
       !_print_ascii_file("/etc/angstrom-version", st)) {
-      st->print("Linux");
+      st->print("Bsd");
   }
   st->cr();

@@ -2025,6 +2246,7 @@
   st->print(name.machine);
   st->cr();

+#ifndef _ALLBSD_SOURCE
   // Print warning if unsafe chroot environment detected
   if (unsafe_chroot_detected) {
     st->print("WARNING!! ");
@@ -2033,12 +2255,13 @@

   // libc, pthread
   st->print("libc:");
-  st->print(os::Linux::glibc_version()); st->print(" ");
-  st->print(os::Linux::libpthread_version()); st->print(" ");
-  if (os::Linux::is_LinuxThreads()) {
-     st->print("(%s stack)", os::Linux::is_floating_stack() ? "floating" : "fixed");
+  st->print(os::Bsd::glibc_version()); st->print(" ");
+  st->print(os::Bsd::libpthread_version()); st->print(" ");
+  if (os::Bsd::is_BsdThreads()) {
+     st->print("(%s stack)", os::Bsd::is_floating_stack() ? "floating" : "fixed");
   }
   st->cr();
+#endif

   // rlimit
   st->print("rlimit:");
@@ -2064,6 +2287,7 @@
   if (rlim.rlim_cur == RLIM_INFINITY) st->print("infinity");
```

```
   else st->print("%d", rlim.rlim_cur);

+#ifndef _ALLBSD_SOURCE
   st->print(", AS ");
   getrlimit(RLIMIT_AS, &rlim);
   if (rlim.rlim_cur == RLIM_INFINITY) st->print("infinity");
@@ -2076,11 +2300,7 @@
   os::loadavg(loadavg, 3);
   st->print("%0.02f %0.02f %0.02f", loadavg[0], loadavg[1], loadavg[2]);
   st->cr();
-
-  // meminfo
-  st->print("\n/proc/meminfo:\n");
-  _print_ascii_file("/proc/meminfo", st);
-  st->cr();
++#endif
 }

 void os::print_memory_info(outputStream* st) {
@@ -2088,23 +2308,32 @@
   st->print("Memory:");
   st->print(" %dk page", os::vm_page_size()>>10);

+#ifndef _ALLBSD_SOURCE
   // values in struct sysinfo are "unsigned long"
   struct sysinfo si;
   sysinfo(&si);
+#endif

   st->print(", physical " UINT64_FORMAT "k",
             os::physical_memory() >> 10);
   st->print("(" UINT64_FORMAT "k free)",
             os::available_memory() >> 10);
+#ifndef _ALLBSD_SOURCE
   st->print(", swap " UINT64_FORMAT "k",
             ((jlong)si.totalswap * si.mem_unit) >> 10);
   st->print("(" UINT64_FORMAT "k free)",
             ((jlong)si.freeswap * si.mem_unit) >> 10);
+#endif
+  st->cr();
+
+  // meminfo
+  st->print("\n/proc/meminfo:\n");
+  _print_ascii_file("/proc/meminfo", st);
   st->cr();
 }

 // Taken from /usr/include/bits/siginfo.h  Supposed to be architecture specific
-// but they're the same for all the linux arch that we support
+// but they're the same for all the bsd arch that we support
 // and they're the same for solaris but there's no common place to put this.
 const char *ill_names[] = { "ILL0", "ILL_ILLOPC", "ILL_ILLOPN", "ILL_ILLADR",
                             "ILL_ILLTRP", "ILL_PRVOPC", "ILL_PRVREG",
@@ -2339,20 +2568,30 @@
 // a counter for each possible signal value
 static volatile jint pending_signals[NSIG+1] = { 0 };

-// Linux(POSIX) specific hand shaking semaphore.
+// Bsd(POSIX) specific hand shaking semaphore.
+#ifdef __APPLE__
+static semaphore_t sig_sem;
+#define SEM_INIT(sem, value)        semaphore_create(mach_task_self(), &sem, SYNC_POLICY_FIFO, value)
+#define SEM_WAIT(sem)               semaphore_wait(sem);
+#define SEM_POST(sem)               semaphore_signal(sem);
+#else
 static sem_t sig_sem;
+#define SEM_INIT(sem, value)        sem_init(&sem, 0, value)
+#define SEM_WAIT(sem)               sem_wait(&sem);
+#define SEM_POST(sem)               sem_post(&sem);
+#endif
```

```
 void os::signal_init_pd() {
   // Initialize signal structures
   ::memset((void*)pending_signals, 0, sizeof(pending_signals));

   // Initialize signal semaphore
-  ::sem_init(&sig_sem, 0, 0);
+  ::SEM_INIT(sig_sem, 0);
 }

 void os::signal_notify(int sig) {
   Atomic::inc(&pending_signals[sig]);
-  ::sem_post(&sig_sem);
+  ::SEM_POST(sig_sem);
 }

 static int check_pending_signals(bool wait) {
@@ -2374,7 +2613,7 @@
     do {
       thread->set_suspend_equivalent();
       // cleared by handle_special_suspend_equivalent_condition() or java_suspend_self()
-      ::sem_wait(&sig_sem);
+      ::SEM_WAIT(sig_sem);

       // were we externally suspended while we were waiting?
       threadIsSuspended = thread->handle_special_suspend_equivalent_condition();
@@ -2385,7 +2624,7 @@
         // while suspended because that would surprise the thread that
         // suspended us.
         //
-        ::sem_post(&sig_sem);
+        ::SEM_POST(sig_sem);

         thread->java_suspend_self();
       }
@@ -2406,14 +2645,14 @@

 int os::vm_page_size() {
   // Seems redundant as all get out
-  assert(os::Linux::page_size() != -1, "must call os::init");
-  return os::Linux::page_size();
+  assert(os::Bsd::page_size() != -1, "must call os::init");
+  return os::Bsd::page_size();
 }

 // Solaris allocates memory by pages.
 int os::vm_allocation_granularity() {
-  assert(os::Linux::page_size() != -1, "must call os::init");
-  return os::Linux::page_size();
+  assert(os::Bsd::page_size() != -1, "must call os::init");
+  return os::Bsd::page_size();
 }

 // Rationale behind this function:
@@ -2422,17 +2661,17 @@
 //  samples for JITted code. Here we create private executable mapping over the code cache
 //  and then we can use standard (well, almost, as mapping can change) way to provide
 //  info for the reporting script by storing timestamp and location of symbol
-void linux_wrap_code(char* base, size_t size) {
+void bsd_wrap_code(char* base, size_t size) {
   static volatile jint cnt = 0;

   if (!UseOprofile) {
     return;
   }

-  char buf[PATH_MAX+1];
+  char buf[PATH_MAX + 1];
   int num = Atomic::add(1, &cnt);

-  snprintf(buf, sizeof(buf), "%s/hs-vm-%d-%d",
+  snprintf(buf, PATH_MAX + 1, "%s/hs-vm-%d-%d",
```

```
            os::get_temp_directory(), os::current_process_id(), num);
   unlink(buf);

@@ -2452,17 +2691,23 @@
   }
 }

-// NOTE: Linux kernel does not really reserve the pages for us.
+// NOTE: Bsd kernel does not really reserve the pages for us.
 //       All it does is to check if there are enough free pages
 //       left at the time of mmap(). This could be a potential
 //       problem.
 bool os::commit_memory(char* addr, size_t size, bool exec) {
   int prot = exec ? PROT_READ|PROT_WRITE|PROT_EXEC : PROT_READ|PROT_WRITE;
+#ifdef __OpenBSD__
+  // XXX: Work-around mmap/MAP_FIXED bug temporarily on OpenBSD
+  return ::mprotect(addr, size, prot) == 0;
+#else
   uintptr_t res = (uintptr_t) ::mmap(addr, size, prot,
                                      MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0);
   return res != (uintptr_t) MAP_FAILED;
+#endif
 }

+#ifndef _ALLBSD_SOURCE
 // Define MAP_HUGETLB here so we can build HotSpot on old systems.
 #ifndef MAP_HUGETLB
 #define MAP_HUGETLB 0x40000
@@ -2472,9 +2717,11 @@
 #ifndef MADV_HUGEPAGE
 #define MADV_HUGEPAGE 14
 #endif
+#endif

 bool os::commit_memory(char* addr, size_t size, size_t alignment_hint,
                        bool exec) {
+#ifndef _ALLBSD_SOURCE
   if (UseHugeTLBFS && alignment_hint > (size_t)vm_page_size()) {
     int prot = exec ? PROT_READ|PROT_WRITE|PROT_EXEC : PROT_READ|PROT_WRITE;
     uintptr_t res =
@@ -2483,16 +2730,19 @@
                            -1, 0);
     return res != (uintptr_t) MAP_FAILED;
   }
+#endif

   return commit_memory(addr, size, exec);
 }

 void os::realign_memory(char *addr, size_t bytes, size_t alignment_hint) {
+#ifndef _ALLBSD_SOURCE
   if (UseHugeTLBFS && alignment_hint > (size_t)vm_page_size()) {
     // We don't check the return value: madvise(MADV_HUGEPAGE) may not
     // be supported or the memory may already be backed by huge pages.
     ::madvise(addr, bytes, MADV_HUGEPAGE);
   }
+#endif
 }

 void os::free_memory(char *addr, size_t bytes) {
@@ -2500,36 +2750,27 @@
 }

 void os::numa_make_global(char *addr, size_t bytes) {
-  Linux::numa_interleave_memory(addr, bytes);
 }

 void os::numa_make_local(char *addr, size_t bytes, int lgrp_hint) {
-  Linux::numa_tonode_memory(addr, bytes, lgrp_hint);
 }
```

```
 bool os::numa_topology_changed()   { return false; }

 size_t os::numa_get_groups_num() {
-  int max_node = Linux::numa_max_node();
-  return max_node > 0 ? max_node + 1 : 1;
+  return 1;
 }

 int os::numa_get_group_id() {
-  int cpu_id = Linux::sched_getcpu();
-  if (cpu_id != -1) {
-    int lgrp_id = Linux::get_node_by_cpu(cpu_id);
-    if (lgrp_id != -1) {
-      return lgrp_id;
-    }
-  }
   return 0;
 }

 size_t os::numa_get_leaf_groups(int *ids, size_t size) {
-  for (size_t i = 0; i < size; i++) {
-    ids[i] = i;
+  if (size > 0) {
+    ids[0] = 0;
+    return 1;
   }
-  return size;
+  return 0;
 }

 bool os::get_page_info(char *start, page_info* info) {
@@ -2540,6 +2781,7 @@
   return end;
 }

+#ifndef _ALLBSD_SOURCE
 // Something to do with the numa-aware allocator needs these symbols
 extern "C" JNIEXPORT void numa_warn(int number, char *where, ...) { }
 extern "C" JNIEXPORT void numa_error(char *where) { }
@@ -2550,7 +2792,7 @@
 // be trying to use symbols with versions 1.1
 // If we are running with earlier version, which did not have symbol versions,
 // we should use the base version.
-void* os::Linux::libnuma_dlsym(void* handle, const char *name) {
+void* os::Bsd::libnuma_dlsym(void* handle, const char *name) {
   void *f = dlvsym(handle, name, "libnuma_1.1");
   if (f == NULL) {
     f = dlsym(handle, name);
@@ -2558,7 +2800,7 @@
   return f;
 }

-bool os::Linux::libnuma_init() {
+bool os::Bsd::libnuma_init() {
   // sched_getcpu() should be in libc.
   set_sched_getcpu(CAST_TO_FN_PTR(sched_getcpu_func_t,
                                   dlsym(RTLD_DEFAULT, "sched_getcpu")));
@@ -2592,7 +2834,7 @@

 // rebuild_cpu_to_node_map() constructs a table mapping cpud id to node id.
 // The table is later used in get_node_by_cpu().
-void os::Linux::rebuild_cpu_to_node_map() {
+void os::Bsd::rebuild_cpu_to_node_map() {
   const size_t NCPUS = 32768; // Since the buffer size computation is very obscure
                               // in libnuma (possible values are starting from 16,
                               // and continuing up with every other power of 2, but less
@@ -2628,113 +2870,41 @@
   FREE_C_HEAP_ARRAY(unsigned long, cpu_map);
 }

-int os::Linux::get_node_by_cpu(int cpu_id) {
```

```
+int os::Bsd::get_node_by_cpu(int cpu_id) {
   if (cpu_to_node() != NULL && cpu_id >= 0 && cpu_id < cpu_to_node()->length()) {
     return cpu_to_node()->at(cpu_id);
   }
   return -1;
 }

-GrowableArray<int>* os::Linux::_cpu_to_node;
-os::Linux::sched_getcpu_func_t os::Linux::_sched_getcpu;
-os::Linux::numa_node_to_cpus_func_t os::Linux::_numa_node_to_cpus;
-os::Linux::numa_max_node_func_t os::Linux::_numa_max_node;
-os::Linux::numa_available_func_t os::Linux::_numa_available;
-os::Linux::numa_tonode_memory_func_t os::Linux::_numa_tonode_memory;
-os::Linux::numa_interleave_memory_func_t os::Linux::_numa_interleave_memory;
-unsigned long* os::Linux::_numa_all_nodes;
+GrowableArray<int>* os::Bsd::_cpu_to_node;
+os::Bsd::sched_getcpu_func_t os::Bsd::_sched_getcpu;
+os::Bsd::numa_node_to_cpus_func_t os::Bsd::_numa_node_to_cpus;
+os::Bsd::numa_max_node_func_t os::Bsd::_numa_max_node;
+os::Bsd::numa_available_func_t os::Bsd::_numa_available;
+os::Bsd::numa_tonode_memory_func_t os::Bsd::_numa_tonode_memory;
+os::Bsd::numa_interleave_memory_func_t os::Bsd::_numa_interleave_memory;
+unsigned long* os::Bsd::_numa_all_nodes;
+#endif

 bool os::uncommit_memory(char* addr, size_t size) {
+#ifdef __OpenBSD__
+  // XXX: Work-around mmap/MAP_FIXED bug temporarily on OpenBSD
+  return ::mprotect(addr, size, PROT_NONE) == 0;
+#else
   uintptr_t res = (uintptr_t) ::mmap(addr, size, PROT_NONE,
                 MAP_PRIVATE|MAP_FIXED|MAP_NORESERVE|MAP_ANONYMOUS, -1, 0);
   return res  != (uintptr_t) MAP_FAILED;
+#endif
 }

-// Linux uses a growable mapping for the stack, and if the mapping for
-// the stack guard pages is not removed when we detach a thread the
-// stack cannot grow beyond the pages where the stack guard was
-// mapped.  If at some point later in the process the stack expands to
-// that point, the Linux kernel cannot expand the stack any further
-// because the guard pages are in the way, and a segfault occurs.
-//
-// However, it's essential not to split the stack region by unmapping
-// a region (leaving a hole) that's already part of the stack mapping,
-// so if the stack mapping has already grown beyond the guard pages at
-// the time we create them, we have to truncate the stack mapping.
-// So, we need to know the extent of the stack mapping when
-// create_stack_guard_pages() is called.
-
-// Find the bounds of the stack mapping.  Return true for success.
-//
-// We only need this for stacks that are growable: at the time of
-// writing thread stacks don't use growable mappings (i.e. those
-// creeated with MAP_GROWSDOWN), and aren't marked "[stack]", so this
-// only applies to the main thread.
-
-static
-bool get_stack_bounds(uintptr_t *bottom, uintptr_t *top) {
-
-  char buf[128];
-  int fd, sz;
-
-  if ((fd = ::open("/proc/self/maps", O_RDONLY)) < 0) {
-    return false;
-  }
-
-  const char kw[] = "[stack]";
-  const int kwlen = sizeof(kw)-1;
-
-  // Address part of /proc/self/maps couldn't be more than 128 bytes
```

```
-  while ((sz = os::get_line_chars(fd, buf, sizeof(buf))) > 0) {
-     if (sz > kwlen && ::memcmp(buf+sz-kwlen, kw, kwlen) == 0) {
-        // Extract addresses
-        if (sscanf(buf, "%" SCNxPTR "-%" SCNxPTR, bottom, top) == 2) {
-           uintptr_t sp = (uintptr_t) __builtin_frame_address(0);
-           if (sp >= *bottom && sp <= *top) {
-              ::close(fd);
-              return true;
-           }
-        }
-     }
-  }
-
- ::close(fd);
-  return false;
-}
-
-
-// If the (growable) stack mapping already extends beyond the point
-// where we're going to put our guard pages, truncate the mapping at
-// that point by munmap()ping it.  This ensures that when we later
-// munmap() the guard pages we don't leave a hole in the stack
-// mapping. This only affects the main/initial thread, but guard
-// against future OS changes
 bool os::create_stack_guard_pages(char* addr, size_t size) {
-  uintptr_t stack_extent, stack_base;
-  bool chk_bounds = NOT_DEBUG(os::Linux::is_initial_thread()) DEBUG_ONLY(true);
-  if (chk_bounds && get_stack_bounds(&stack_extent, &stack_base)) {
-      assert(os::Linux::is_initial_thread(),
-           "growable stack in non-initial thread");
-    if (stack_extent < (uintptr_t)addr)
-      ::munmap((void*)stack_extent, (uintptr_t)addr - stack_extent);
-  }
-
    return os::commit_memory(addr, size);
 }

 // If this is a growable mapping, remove the guard pages entirely by
-// munmap()ping them.  If not, just call uncommit_memory(). This only
-// affects the main/initial thread, but guard against future OS changes
+// munmap()ping them.  If not, just call uncommit_memory().
 bool os::remove_stack_guard_pages(char* addr, size_t size) {
-  uintptr_t stack_extent, stack_base;
-  bool chk_bounds = NOT_DEBUG(os::Linux::is_initial_thread()) DEBUG_ONLY(true);
-  if (chk_bounds && get_stack_bounds(&stack_extent, &stack_base)) {
-      assert(os::Linux::is_initial_thread(),
-           "growable stack in non-initial thread");
-
-    return ::munmap(addr, size) == 0;
-  }
-
    return os::uncommit_memory(addr, size);
 }

@@ -2744,7 +2914,7 @@
 // at 'requested_addr'. If there are existing memory mappings at the same
 // location, however, they will be overwritten. If 'fixed' is false,
 // 'requested_addr' is only treated as a hint, the return value may or
-// may not start from the requested address. Unlike Linux mmap(), this
+// may not start from the requested address. Unlike Bsd mmap(), this
 // function returns NULL to indicate failure.
 static char* anon_mmap(char* requested_addr, size_t bytes, bool fixed) {
   char * addr;
@@ -2752,7 +2922,7 @@

   flags = MAP_PRIVATE | MAP_NORESERVE | MAP_ANONYMOUS;
   if (fixed) {
-    assert((uintptr_t)requested_addr % os::Linux::page_size() == 0, "unaligned address");
+    assert((uintptr_t)requested_addr % os::Bsd::page_size() == 0, "unaligned address");
     flags |= MAP_FIXED;
   }
```

```
@@ -2795,9 +2965,9 @@
   return _highest_vm_reserved_address;
 }

-static bool linux_mprotect(char* addr, size_t size, int prot) {
-  // Linux wants the mprotect address argument to be page aligned.
-  char* bottom = (char*)align_size_down((intptr_t)addr, os::Linux::page_size());
+static bool bsd_mprotect(char* addr, size_t size, int prot) {
+  // Bsd wants the mprotect address argument to be page aligned.
+  char* bottom = (char*)align_size_down((intptr_t)addr, os::Bsd::page_size());

   // According to SUSv3, mprotect() should only be used with mappings
   // established by mmap(), and mmap() always maps whole pages. Unaligned
@@ -2806,7 +2976,7 @@
   // caller if you hit this assert.
   assert(addr == bottom, "sanity check");

-  size = align_size_up(pointer_delta(addr, bottom, 1) + size, os::Linux::page_size());
+  size = align_size_up(pointer_delta(addr, bottom, 1) + size, os::Bsd::page_size());
   return ::mprotect(bottom, size, prot) == 0;
 }

@@ -2823,19 +2993,20 @@
     ShouldNotReachHere();
   }
   // is_committed is unused.
-  return linux_mprotect(addr, bytes, p);
+  return bsd_mprotect(addr, bytes, p);
 }

 bool os::guard_memory(char* addr, size_t size) {
-  return linux_mprotect(addr, size, PROT_NONE);
+  return bsd_mprotect(addr, size, PROT_NONE);
 }

 bool os::unguard_memory(char* addr, size_t size) {
-  return linux_mprotect(addr, size, PROT_READ|PROT_WRITE);
+  return bsd_mprotect(addr, size, PROT_READ|PROT_WRITE);
 }

-bool os::Linux::hugetlbfs_sanity_check(bool warn, size_t page_size) {
+bool os::Bsd::hugetlbfs_sanity_check(bool warn, size_t page_size) {
   bool result = false;
+#ifndef _ALLBSD_SOURCE
   void *p = mmap (NULL, page_size, PROT_READ|PROT_WRITE,
                   MAP_ANONYMOUS|MAP_PRIVATE|MAP_HUGETLB,
                   -1, 0);
@@ -2867,6 +3038,7 @@
   if (warn) {
     warning("HugeTLBFS is not supported by the operating system.");
   }
+#endif

   return result;
 }
@@ -2913,6 +3085,7 @@
 static size_t _large_page_size = 0;

 void os::large_page_init() {
+#ifndef _ALLBSD_SOURCE
   if (!UseLargePages) {
     UseHugeTLBFS = false;
     UseSHM = false;
@@ -2932,7 +3105,7 @@
   if (LargePageSizeInBytes) {
     _large_page_size = LargePageSizeInBytes;
   } else {
-    // large_page_size on Linux is used to round up heap size. x86 uses either
+    // large_page_size on Bsd is used to round up heap size. x86 uses either
     // 2M or 4M page, depending on whether PAE (Physical Address Extensions)
```

```
     // mode is enabled. AMD64/EM64T uses 2M page in 64bit mode. IA64 can use
     // page as large as 256M.
@@ -2975,14 +3148,14 @@
   // print a warning if any large page related flag is specified on command line
   bool warn_on_failure = !FLAG_IS_DEFAULT(UseHugeTLBFS);

-  const size_t default_page_size = (size_t)Linux::page_size();
+  const size_t default_page_size = (size_t)Bsd::page_size();
   if (_large_page_size > default_page_size) {
     _page_sizes[0] = _large_page_size;
     _page_sizes[1] = default_page_size;
     _page_sizes[2] = 0;
   }
   UseHugeTLBFS = UseHugeTLBFS &&
-                 Linux::hugetlbfs_sanity_check(warn_on_failure, _large_page_size);
+                 Bsd::hugetlbfs_sanity_check(warn_on_failure, _large_page_size);

   if (UseHugeTLBFS)
     UseSHM = false;
@@ -2990,11 +3163,14 @@
   UseLargePages = UseHugeTLBFS || UseSHM;

   set_coredump_filter();
+#endif
 }

+#ifndef _ALLBSD_SOURCE
 #ifndef SHM_HUGETLB
 #define SHM_HUGETLB 04000
 #endif
+#endif

 char* os::reserve_memory_special(size_t bytes, char* req_addr, bool exec) {
   // "exec" is passed in but not used.  Creating the shared image for
@@ -3012,7 +3188,11 @@

   // Create a large shared memory region to attach to based on size.
   // Currently, size is the total size of the heap
+#ifndef _ALLBSD_SOURCE
   int shmid = shmget(key, bytes, SHM_HUGETLB|IPC_CREAT|SHM_R|SHM_W);
+#else
+  int shmid = shmget(key, bytes, IPC_CREAT|SHM_R|SHM_W);
+#endif
   if (shmid == -1) {
     // Possible reasons for shmget failure:
     // 1. shmmax is too small for Java heap.
@@ -3022,7 +3202,7 @@
     //    > check available large pages: cat /proc/meminfo
     //    > increase amount of large pages:
     //          echo new_value > /proc/sys/vm/nr_hugepages
-    //      Note 1: different Linux may use different name for this property,
+    //      Note 1: different Bsd may use different name for this property,
     //            e.g. on Redhat AS-3 it is "hugetlb_pool".
     //      Note 2: it's possible there's enough physical memory available but
     //            they are so fragmented after a long run that they can't
@@ -3098,13 +3278,13 @@
   // automatically update _highest_vm_reserved_address if the call is
   // successful. The variable tracks the highest memory address every reserved
   // by JVM. It is used to detect heap-stack collision if running with
-  // fixed-stack LinuxThreads. Because here we may attempt to reserve more
+  // fixed-stack BsdThreads. Because here we may attempt to reserve more
   // space than needed, it could confuse the collision detecting code. To
   // solve the problem, save current _highest_vm_reserved_address and
   // calculate the correct value before return.
   address old_highest = _highest_vm_reserved_address;

-  // Linux mmap allows caller to pass an address as hint; give it a try first,
+  // Bsd mmap allows caller to pass an address as hint; give it a try first,
   // if kernel honors the hint then we can return immediately.
   char * addr = anon_mmap(requested_addr, bytes, false);
   if (addr == requested_addr) {
```

```
@@ -3165,11 +3345,11 @@
 }

 size_t os::read(int fd, void *buf, unsigned int nBytes) {
-  return ::read(fd, buf, nBytes);
+  RESTARTABLE_RETURN_INT(::read(fd, buf, nBytes));
 }

-// TODO-FIXME: reconcile Solaris' os::sleep with the linux variation.
-// Solaris uses poll(), linux uses park().
+// TODO-FIXME: reconcile Solaris' os::sleep with the bsd variation.
+// Solaris uses poll(), bsd uses park().
 // Poll() is likely a better choice, assuming that Thread.interrupt()
 // generates a SIGUSRx signal. Note that SIGUSR1 can interfere with
 // SIGSEGV, see 4355769.
@@ -3196,7 +3376,7 @@
       if (newtime - prevtime < 0) {
         // time moving backwards, should only happen if no monotonic clock
         // not a guarantee() because JVM should not abort on kernel/glibc bugs
-        assert(!Linux::supports_monotonic_clock(), "time moving backwards");
+        assert(!Bsd::supports_monotonic_clock(), "time moving backwards");
       } else {
         millis -= (newtime - prevtime) / NANOSECS_PER_MILLISECS;
       }
@@ -3235,7 +3415,7 @@
       if (newtime - prevtime < 0) {
         // time moving backwards, should only happen if no monotonic clock
         // not a guarantee() because JVM should not abort on kernel/glibc bugs
-        assert(!Linux::supports_monotonic_clock(), "time moving backwards");
+        assert(!Bsd::supports_monotonic_clock(), "time moving backwards");
       } else {
         millis -= (newtime - prevtime) / NANOSECS_PER_MILLISECS;
       }
@@ -3274,7 +3454,7 @@

 void os::yield_all(int attempts) {
   // Yields to all threads, including threads with lower priorities
-  // Threads on Linux are all with same priority. The Solaris style
+  // Threads on Bsd are all with same priority. The Solaris style
   // os::yield_all() with nanosleep(1ms) is not necessary.
   sched_yield();
 }
@@ -3287,9 +3467,9 @@
 ////////////////////////////////////////////////////////////////////////////////
 // thread priority support

-// Note: Normal Linux applications are run with SCHED_OTHER policy. SCHED_OTHER
+// Note: Normal Bsd applications are run with SCHED_OTHER policy. SCHED_OTHER
 // only supports dynamic priority, static priority must be zero. For real-time
-// applications, Linux supports SCHED_RR which allows static priority (1-99).
+// applications, Bsd supports SCHED_RR which allows static priority (1-99).
 // However, for large multi-threaded applications, SCHED_RR is not only slower
 // than SCHED_OTHER, but also very unstable (my volano tests hang hard 4 out
 // of 5 runs - Sep 2005).
@@ -3301,6 +3481,44 @@
 // this reason, the code should not be used as default (ThreadPriorityPolicy=0).
 // It is only used when ThreadPriorityPolicy=1 and requires root privilege.

+#if defined(_ALLBSD_SOURCE) && !defined(__APPLE__)
+int os::java_to_os_priority[MaxPriority + 1] = {
+  19,              // 0 Entry should never be used
+
+   0,              // 1 MinPriority
+   3,              // 2
+   6,              // 3
+
+   10,              // 4
+   15,              // 5 NormPriority
+   18,              // 6
+
+   21,              // 7
```

```
+   25,              // 8
+   28,              // 9 NearMaxPriority
+
+   31              // 10 MaxPriority
+};
+#elif defined(__APPLE__)
+/* Using Mach high-level priority assignments */
+int os::java_to_os_priority[MaxPriority + 1] = {
+   0,              // 0 Entry should never be used (MINPRI_USER)
+
+  27,              // 1 MinPriority
+  28,              // 2
+  29,              // 3
+
+  30,              // 4
+  31,              // 5 NormPriority (BASEPRI_DEFAULT)
+  32,              // 6
+
+  33,              // 7
+  34,              // 8
+  35,              // 9 NearMaxPriority
+
+  36               // 10 MaxPriority
+};
+#else
 int os::java_to_os_priority[MaxPriority + 1] = {
   19,              // 0 Entry should never be used

@@ -3318,6 +3536,7 @@

   -5               // 10 MaxPriority
 };
+#endif

 static int prio_init() {
   if (ThreadPriorityPolicy == 1) {
@@ -3326,7 +3545,7 @@
     // this is to test CAP_SYS_NICE capability, but that will require libcap.so
     if (geteuid() != 0) {
       if (!FLAG_IS_DEFAULT(ThreadPriorityPolicy)) {
-        warning("-XX:ThreadPriorityPolicy requires root privilege on Linux");
+        warning("-XX:ThreadPriorityPolicy requires root privilege on Bsd");
       }
       ThreadPriorityPolicy = 0;
     }
@@ -3337,8 +3556,28 @@
 OSReturn os::set_native_priority(Thread* thread, int newpri) {
   if ( !UseThreadPriorities || ThreadPriorityPolicy == 0 ) return OS_OK;

+#ifdef __OpenBSD__
+  // OpenBSD pthread_setprio starves low priority threads
+  return OS_OK;
+#elif defined(__FreeBSD__)
+  int ret = pthread_setprio(thread->osthread()->pthread_id(), newpri);
+#elif defined(__APPLE__) || defined(__NetBSD__)
+  struct sched_param sp;
+  int policy;
+  pthread_t self = pthread_self();
+
+  if (pthread_getschedparam(self, &policy, &sp) != 0)
+    return OS_ERR;
+
+  sp.sched_priority = newpri;
+  if (pthread_setschedparam(self, policy, &sp) != 0)
+    return OS_ERR;
+
+  return OS_OK;
+#else
   int ret = setpriority(PRIO_PROCESS, thread->osthread()->thread_id(), newpri);
   return (ret == 0) ? OS_OK : OS_ERR;
+#endif
```

```
 }

 OSReturn os::get_native_priority(const Thread* const thread, int *priority_ptr) {
@@ -3348,7 +3587,17 @@
   }

   errno = 0;
+#if defined(__OpenBSD__) || defined(__FreeBSD__)
+  *priority_ptr = pthread_getprio(thread->osthread()->pthread_id());
+#elif defined(__APPLE__) || defined(__NetBSD__)
+  int policy;
+  struct sched_param sp;
+
+  pthread_getschedparam(pthread_self(), &policy, &sp);
+  *priority_ptr = sp.sched_priority;
+#else
   *priority_ptr = getpriority(PRIO_PROCESS, thread->osthread()->thread_id());
+#endif
   return (*priority_ptr != -1 || errno == 0 ? OS_OK : OS_ERR);
 }

@@ -3458,7 +3707,7 @@
   /* Get signal number to use for suspend/resume */
   if ((s = ::getenv("_JAVA_SR_SIGNUM")) != 0) {
     int sig = ::strtol(s, 0, 10);
-    if (sig > 0 || sig < _NSIG) {
+    if (sig > 0 || sig < NSIG) {
         SR_signum = sig;
     }
   }
@@ -3475,7 +3724,7 @@

   // SR_signum is blocked by default.
   // 4528190 - We also need to block pthread restart signal (32 on all
-  // supported Linux platforms). Note that LinuxThreads need to block
+  // supported Bsd platforms). Note that BsdThreads need to block
   // this signal for all threads to work properly. So we don't have
   // to use hard-coded signal number when setting up the mask.
   pthread_sigmask(SIG_BLOCK, NULL, &act.sa_mask);
@@ -3485,7 +3734,7 @@
   }

   // Save signal flag
-  os::Linux::set_our_sigflags(SR_signum, act.sa_flags);
+  os::Bsd::set_our_sigflags(SR_signum, act.sa_flags);
   return 0;
 }

@@ -3604,27 +3853,27 @@
 // handlers, unless invoked with the option "-XX:+AllowUserSignalHandlers".
 //
 extern "C" JNIEXPORT int
-JVM_handle_linux_signal(int signo, siginfo_t* siginfo,
+JVM_handle_bsd_signal(int signo, siginfo_t* siginfo,
                         void* ucontext, int abort_if_unrecognized);

 void signalHandler(int sig, siginfo_t* info, void* uc) {
   assert(info != NULL && uc != NULL, "it must be old kernel");
-  JVM_handle_linux_signal(sig, info, uc, true);
+  JVM_handle_bsd_signal(sig, info, uc, true);
 }


 // This boolean allows users to forward their own non-matching signals
-// to JVM_handle_linux_signal, harmlessly.
-bool os::Linux::signal_handlers_are_installed = false;
+// to JVM_handle_bsd_signal, harmlessly.
+bool os::Bsd::signal_handlers_are_installed = false;

 // For signal-chaining
-struct sigaction os::Linux::sigact[MAXSIGNUM];
```

```
-unsigned int os::Linux::sigs = 0;
-bool os::Linux::libjsig_is_loaded = false;
+struct sigaction os::Bsd::sigact[MAXSIGNUM];
+unsigned int os::Bsd::sigs = 0;
+bool os::Bsd::libjsig_is_loaded = false;
 typedef struct sigaction *(*get_signal_t)(int);
-get_signal_t os::Linux::get_signal_action = NULL;
+get_signal_t os::Bsd::get_signal_action = NULL;

-struct sigaction* os::Linux::get_chained_signal_action(int sig) {
+struct sigaction* os::Bsd::get_chained_signal_action(int sig) {
   struct sigaction *actp = NULL;

   if (libjsig_is_loaded) {
@@ -3684,7 +3933,7 @@
   return true;
 }

-bool os::Linux::chained_handler(int sig, siginfo_t* siginfo, void* context) {
+bool os::Bsd::chained_handler(int sig, siginfo_t* siginfo, void* context) {
   bool chained = false;
   // signal-chaining
   if (UseSignalChaining) {
@@ -3696,33 +3945,33 @@
   return chained;
 }

-struct sigaction* os::Linux::get_preinstalled_handler(int sig) {
+struct sigaction* os::Bsd::get_preinstalled_handler(int sig) {
   if ((( (unsigned int)1 << sig ) & sigs) != 0) {
     return &sigact[sig];
   }
   return NULL;
 }

-void os::Linux::save_preinstalled_handler(int sig, struct sigaction& oldAct) {
+void os::Bsd::save_preinstalled_handler(int sig, struct sigaction& oldAct) {
   assert(sig > 0 && sig < MAXSIGNUM, "vm signal out of expected range");
   sigact[sig] = oldAct;
   sigs |= (unsigned int)1 << sig;
 }

 // for diagnostic
-int os::Linux::sigflags[MAXSIGNUM];
+int os::Bsd::sigflags[MAXSIGNUM];

-int os::Linux::get_our_sigflags(int sig) {
+int os::Bsd::get_our_sigflags(int sig) {
   assert(sig > 0 && sig < MAXSIGNUM, "vm signal out of expected range");
   return sigflags[sig];
 }

-void os::Linux::set_our_sigflags(int sig, int flags) {
+void os::Bsd::set_our_sigflags(int sig, int flags) {
   assert(sig > 0 && sig < MAXSIGNUM, "vm signal out of expected range");
   sigflags[sig] = flags;
 }

-void os::Linux::set_signal_handler(int sig, bool set_installed) {
+void os::Bsd::set_signal_handler(int sig, bool set_installed) {
   // Check for overwrite.
   struct sigaction oldAct;
   sigaction(sig, (struct sigaction*)NULL, &oldAct);
@@ -3772,7 +4021,7 @@
 // install signal handlers for signals that HotSpot needs to
 // handle in order to support Java-level exception handling.

-void os::Linux::install_signal_handlers() {
+void os::Bsd::install_signal_handlers() {
   if (!signal_handlers_are_installed) {
     signal_handlers_are_installed = true;
```

```
@@ -3802,6 +4051,28 @@
     set_signal_handler(SIGFPE, true);
     set_signal_handler(SIGXFSZ, true);

+#if defined(__APPLE__)
+    // In Mac OS X 10.4, CrashReporter will write a crash log for all 'fatal' signals, including
+    // signals caught and handled by the JVM. To work around this, we reset the mach task
+    // signal handler that's placed on our process by CrashReporter. This disables
+    // CrashReporter-based reporting.
+    //
+    // This work-around is not necessary for 10.5+, as CrashReporter no longer intercedes
+    // on caught fatal signals.
+    //
+    // Additionally, gdb installs both standard BSD signal handlers, and mach exception
+    // handlers. By replacing the existing task exception handler, we disable gdb's mach
+    // exception handling, while leaving the standard BSD signal handlers functional.
+    kern_return_t kr;
+    kr = task_set_exception_ports(mach_task_self(),
+        EXC_MASK_BAD_ACCESS | EXC_MASK_ARITHMETIC,
+        MACH_PORT_NULL,
+        EXCEPTION_STATE_IDENTITY,
+        MACHINE_THREAD_STATE);
+
+    assert(kr == KERN_SUCCESS, "could not set mach task signal handler");
+#endif
+
     if (libjsig_is_loaded) {
       // Tell libjsig jvm finishes setting signal handlers
       (*end_signal_setting)();
@@ -3822,23 +4093,25 @@
   }
 }

-// This is the fastest way to get thread cpu time on Linux.
+#ifndef _ALLBSD_SOURCE
+// This is the fastest way to get thread cpu time on Bsd.
 // Returns cpu time (user+sys) for any thread, not only for current.
 // POSIX compliant clocks are implemented in the kernels 2.6.16+.
 // It might work on 2.6.10+ with a special kernel/glibc patch.
 // For reference, please, see IEEE Std 1003.1-2004:
 //   http://www.unix.org/single_unix_specification

-jlong os::Linux::fast_thread_cpu_time(clockid_t clockid) {
+jlong os::Bsd::fast_thread_cpu_time(clockid_t clockid) {
   struct timespec tp;
-  int rc = os::Linux::clock_gettime(clockid, &tp);
+  int rc = os::Bsd::clock_gettime(clockid, &tp);
   assert(rc == 0, "clock_gettime is expected to return 0 code");

   return (tp.tv_sec * SEC_IN_NANOSECS) + tp.tv_nsec;
 }
+#endif

 /////
-// glibc on Linux platform uses non-documented flag
+// glibc on Bsd platform uses non-documented flag
 // to indicate, that some special sort of signal
 // trampoline is used.
 // We will never set this flag, and we should
@@ -3904,10 +4177,10 @@
      handler == CAST_FROM_FN_PTR(address, (sa_sigaction_t)SR_handler)) {
     // It is our signal handler
     // check for flags, reset system-used one!
-    if((int)sa.sa_flags != os::Linux::get_our_sigflags(sig)) {
+    if((int)sa.sa_flags != os::Bsd::get_our_sigflags(sig)) {
       st->print(
                 ", flags was changed from " PTR32_FORMAT ", consider using jsig library",
-                os::Linux::get_our_sigflags(sig));
+                os::Bsd::get_our_sigflags(sig));
     }
```

```
   }
   st->cr();
@@ -3916,7 +4189,7 @@

 #define DO_SIGNAL_CHECK(sig) \
   if (!sigismember(&check_signal_done, sig)) \
-    os::Linux::check_signal_handler(sig)
+    os::Bsd::check_signal_handler(sig)

 // This method is a periodic task to check for misbehaving JNI applications
 // under CheckJNI, we can add any periodic checks here
@@ -3954,7 +4227,7 @@

 static os_sigaction_t os_sigaction = NULL;

-void os::Linux::check_signal_handler(int sig) {
+void os::Bsd::check_signal_handler(int sig) {
   char buf[O_BUFLEN];
   address jvmHandler = NULL;

@@ -4012,9 +4285,9 @@
     tty->print_cr("  found:%s", get_signal_handler_name(thisHandler, buf, O_BUFLEN));
     // No need to check this sig any longer
     sigaddset(&check_signal_done, sig);
-  } else if(os::Linux::get_our_sigflags(sig) != 0 && (int)act.sa_flags != os::Linux::get_our_sigflags(sig)) {
+  } else if(os::Bsd::get_our_sigflags(sig) != 0 && (int)act.sa_flags != os::Bsd::get_our_sigflags(sig)) {
     tty->print("Warning: %s handler flags ", exception_name(sig, buf, O_BUFLEN));
-    tty->print("expected:" PTR32_FORMAT, os::Linux::get_our_sigflags(sig));
+    tty->print("expected:" PTR32_FORMAT, os::Bsd::get_our_sigflags(sig));
     tty->print_cr("  found:" PTR32_FORMAT, act.sa_flags);
     // No need to check this sig any longer
     sigaddset(&check_signal_done, sig);
@@ -4047,9 +4320,9 @@
   char dummy;   /* used to get a guess on initial stack address */
 //  first_hrtime = gethrtime();

-  // With LinuxThreads the JavaMain thread pid (primordial thread)
+  // With BsdThreads the JavaMain thread pid (primordial thread)
   // is different than the pid of the java launcher thread.
-  // So, on Linux, the launcher thread pid is passed to the VM
+  // So, on Bsd, the launcher thread pid is passed to the VM
   // via the sun.java.launcher.pid property.
   // Use this property instead of getpid() if it was correctly passed.
   // See bug 6351349.
@@ -4057,27 +4330,36 @@

   _initial_pid = (java_launcher_pid > 0) ? java_launcher_pid : getpid();

-  clock_tics_per_sec = sysconf(_SC_CLK_TCK);
+  clock_tics_per_sec = CLK_TCK;

   init_random(1234567);

   ThreadCritical::initialize();

-  Linux::set_page_size(sysconf(_SC_PAGESIZE));
-  if (Linux::page_size() == -1) {
-    fatal(err_msg("os_linux.cpp: os::init: sysconf failed (%s)",
+  Bsd::set_page_size(getpagesize());
+  if (Bsd::page_size() == -1) {
+    fatal(err_msg("os_bsd.cpp: os::init: sysconf failed (%s)",
                   strerror(errno)));
   }
-  init_page_sizes((size_t) Linux::page_size());
+  init_page_sizes((size_t) Bsd::page_size());

-  Linux::initialize_system_info();
+  Bsd::initialize_system_info();

   // main_thread points to the aboriginal thread
-  Linux::_main_thread = pthread_self();
```

```
+  Bsd::_main_thread = pthread_self();

-  Linux::clock_init();
+  Bsd::clock_init();
   initial_time_count = os::elapsed_counter();
-  pthread_mutex_init(&dl_mutex, NULL);
+
+#ifdef __APPLE__
+  // XXXDARWIN
+  // Work around the unaligned VM callbacks in hotspot's
+  // sharedRuntime. The callbacks don't use SSE2 instructions, and work on
+  // Linux, Solaris, and FreeBSD. On Mac OS X, dyld (rightly so) enforces
+  // alignment when doing symbol lookup. To work around this, we force early
+  // binding of all symbols now, thus binding when alignment is known-good.
+  _dyld_bind_fully_image_containing_address((const void *) &os::init);
+#endif
 }

 // To install functions for atexit system call
@@ -4090,10 +4372,12 @@
 // this is called _after_ the global arguments have been parsed
 jint os::init_2(void)
 {
-  Linux::fast_thread_clock_init();
+#ifndef _ALLBSD_SOURCE
+  Bsd::fast_thread_clock_init();
+#endif

   // Allocate a single page and mark it as readable for safepoint polling
-  address polling_page = (address) ::mmap(NULL, Linux::page_size(), PROT_READ, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0);
+  address polling_page = (address) ::mmap(NULL, Bsd::page_size(), PROT_READ, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
   guarantee( polling_page != MAP_FAILED, "os::init_2: failed to allocate polling page" );

   os::set_polling_page( polling_page );
@@ -4104,7 +4388,7 @@
 #endif

   if (!UseMembar) {
-    address mem_serialize_page = (address) ::mmap(NULL, Linux::page_size(), PROT_READ | PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
+    address mem_serialize_page = (address) ::mmap(NULL, Bsd::page_size(), PROT_READ | PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
     guarantee( mem_serialize_page != NULL, "mmap Failed for memory serialize page");
     os::set_memory_serialize_page( mem_serialize_page );

@@ -4122,24 +4406,24 @@
     return JNI_ERR;
   }

-  Linux::signal_sets_init();
-  Linux::install_signal_handlers();
+  Bsd::signal_sets_init();
+  Bsd::install_signal_handlers();

   // Check minimum allowable stack size for thread creation and to initialize
   // the java system classes, including StackOverflowError - depends on page
   // size.  Add a page for compiler2 recursion in main thread.
   // Add in 2*BytesPerWord times page size to account for VM stack during
   // class initialization depending on 32 or 64 bit VM.
-  os::Linux::min_stack_allowed = MAX2(os::Linux::min_stack_allowed,
+  os::Bsd::min_stack_allowed = MAX2(os::Bsd::min_stack_allowed,
             (size_t)(StackYellowPages+StackRedPages+StackShadowPages+
-                    2*BytesPerWord COMPILER2_PRESENT(+1)) * Linux::page_size());
+                    2*BytesPerWord COMPILER2_PRESENT(+1)) * Bsd::page_size());

   size_t threadStackSizeInBytes = ThreadStackSize * K;
   if (threadStackSizeInBytes != 0 &&
-      threadStackSizeInBytes < os::Linux::min_stack_allowed) {
+      threadStackSizeInBytes < os::Bsd::min_stack_allowed) {
     tty->print_cr("\nThe stack size specified is too small, "
```

```
                      "Specify at least %dk",
-                     os::Linux::min_stack_allowed/ K);
+                     os::Bsd::min_stack_allowed/ K);
         return JNI_ERR;
   }

@@ -4148,20 +4432,21 @@
   JavaThread::set_stack_size_at_create(round_to(threadStackSizeInBytes,
         vm_page_size()));

-  Linux::capture_initial_stack(JavaThread::stack_size_at_create());
+#ifndef _ALLBSD_SOURCE
+  Bsd::capture_initial_stack(JavaThread::stack_size_at_create());

-  Linux::libpthread_init();
+  Bsd::libpthread_init();
   if (PrintMiscellaneous && (Verbose || WizardMode)) {
      tty->print_cr("[HotSpot is running with %s, %s(%s)]\n",
-          Linux::glibc_version(), Linux::libpthread_version(),
-          Linux::is_floating_stack() ? "floating stack" : "fixed stack");
+          Bsd::glibc_version(), Bsd::libpthread_version(),
+          Bsd::is_floating_stack() ? "floating stack" : "fixed stack");
   }

   if (UseNUMA) {
-    if (!Linux::libnuma_init()) {
+    if (!Bsd::libnuma_init()) {
       UseNUMA = false;
     } else {
-      if ((Linux::numa_max_node() < 1)) {
+      if ((Bsd::numa_max_node() < 1)) {
         // There's only one node(they start from 0), disable NUMA.
         UseNUMA = false;
       }
@@ -4187,6 +4472,7 @@
       UseNUMA = true;
     }
   }
+#endif

   if (MaxFDLimit) {
     // set the number of file descriptors to max. print out error
@@ -4198,6 +4484,14 @@
         perror("os::init_2 getrlimit failed");
     } else {
       nbr_files.rlim_cur = nbr_files.rlim_max;
+
+#ifdef __APPLE__
+      // Darwin returns RLIM_INFINITY for rlim_max, but fails with EINVAL if
+      // you attempt to use RLIM_INFINITY. As per setrlimit(2), OPEN_MAX must
+      // be used instead
+      nbr_files.rlim_cur = MIN(OPEN_MAX, nbr_files.rlim_cur);
+#endif
+
       status = setrlimit(RLIMIT_NOFILE, &nbr_files);
       if (status != 0) {
         if (PrintMiscellaneous && (Verbose || WizardMode))
@@ -4206,8 +4500,10 @@
     }
   }

+#ifndef _ALLBSD_SOURCE
   // Initialize lock used to serialize thread creation (see os::create_thread)
-  Linux::set_createThread_lock(new Mutex(Mutex::leaf, "createThread_lock", false));
+  Bsd::set_createThread_lock(new Mutex(Mutex::leaf, "createThread_lock", false));
+#endif

   // at-exit methods are called in the reverse order of their registration.
   // atexit functions are called on return from main or as a result of a
@@ -4239,23 +4535,27 @@
```

```
 // Mark the polling page as unreadable
 void os::make_polling_page_unreadable(void) {
-  if( !guard_memory((char*)_polling_page, Linux::page_size()) )
+  if( !guard_memory((char*)_polling_page, Bsd::page_size()) )
     fatal("Could not disable polling page");
 };

 // Mark the polling page as readable
 void os::make_polling_page_readable(void) {
-  if( !linux_mprotect((char *)_polling_page, Linux::page_size(), PROT_READ)) {
+  if( !bsd_mprotect((char *)_polling_page, Bsd::page_size(), PROT_READ)) {
     fatal("Could not enable polling page");
   }
 };

 int os::active_processor_count() {
-  // Linux doesn't yet have a (official) notion of processor sets,
+#ifdef _ALLBSD_SOURCE
+  return _processor_count;
+#else
+  // Bsd doesn't yet have a (official) notion of processor sets,
   // so just return the number of online processors.
   int online_cpus = ::sysconf(_SC_NPROCESSORS_ONLN);
   assert(online_cpus > 0 && online_cpus <= processor_count(), "sanity check");
   return online_cpus;
+#endif
 }

 bool os::distribute_processes(uint length, uint* distribution) {
@@ -4282,7 +4582,7 @@
   OSThread* osthread = thread->osthread();
   if (do_suspend(osthread)) {
     if (osthread->ucontext() != NULL) {
-      epc = os::Linux::ucontext_get_pc(osthread->ucontext());
+      epc = os::Bsd::ucontext_get_pc(osthread->ucontext());
     } else {
       // NULL context is unexpected, double-check this is the VMThread
       guarantee(thread->is_VM_thread(), "can only be called for VMThread");
@@ -4295,13 +4595,16 @@
   return epc;
 }

-int os::Linux::safe_cond_timedwait(pthread_cond_t *_cond, pthread_mutex_t *_mutex, const struct timespec
*_abstime)
+int os::Bsd::safe_cond_timedwait(pthread_cond_t *_cond, pthread_mutex_t *_mutex, const struct timespec
*_abstime)
 {
+#ifdef _ALLBSD_SOURCE
+  return pthread_cond_timedwait(_cond, _mutex, _abstime);
+#else
    if (is_NPTL()) {
       return pthread_cond_timedwait(_cond, _mutex, _abstime);
    } else {
 #ifndef IA64
-      // 6292965: LinuxThreads pthread_cond_timedwait() resets FPU control
+      // 6292965: BsdThreads pthread_cond_timedwait() resets FPU control
       // word back to default 64bit precision if condvar is signaled. Java
       // wants 53bit precision.  Save and restore current value.
       int fpu = get_fpu_control_word();
@@ -4312,6 +4615,7 @@
 #endif // IA64
       return status;
    }
+#endif
 }

 ////////////////////////////////////////////////////////////////////////////////
@@ -4369,7 +4673,7 @@
 ////////////////////////////////////////////////////////////////////////////////
 // misc
```

```
-// This does not do anything on Linux. This is basically a hook for being
+// This does not do anything on Bsd. This is basically a hook for being
 // able to use structured exception handling (thread-local exception filters)
 // on, e.g., Win32.
 void
@@ -4459,14 +4763,14 @@
   int o_delete = (oflag & O_DELETE);
   oflag = oflag & ~O_DELETE;

-  fd = ::open64(path, oflag, mode);
+  fd = ::open(path, oflag, mode);
   if (fd == -1) return -1;

   //If the open succeeded, the file might still be a directory
   {
-    struct stat64 buf64;
-    int ret = ::fstat64(fd, &buf64);
-    int st_mode = buf64.st_mode;
+    struct stat buf;
+    int ret = ::fstat(fd, &buf);
+    int st_mode = buf.st_mode;

     if (ret != -1) {
       if ((st_mode & S_IFMT) == S_IFDIR) {
@@ -4523,17 +4827,17 @@
   if (!rewrite_existing) {
     oflags |= O_EXCL;
   }
-  return ::open64(path, oflags, S_IREAD | S_IWRITE);
+  return ::open(path, oflags, S_IREAD | S_IWRITE);
 }

 // return current position of file pointer
 jlong os::current_file_offset(int fd) {
-  return (jlong)::lseek64(fd, (off64_t)0, SEEK_CUR);
+  return (jlong)::lseek(fd, (off_t)0, SEEK_CUR);
 }

 // move file pointer to the specified offset
 jlong os::seek_to_file_offset(int fd, jlong offset) {
-  return (jlong)::lseek64(fd, (off64_t)offset, SEEK_SET);
+  return (jlong)::lseek(fd, (off_t)offset, SEEK_SET);
 }

 // This code originates from JDK's sysAvailable
@@ -4542,10 +4846,10 @@
   jlong cur, end;
   int mode;
-  struct stat64 buf64;
+  struct stat buf;

-  if (::fstat64(fd, &buf64) >= 0) {
-    mode = buf64.st_mode;
+  if (::fstat(fd, &buf) >= 0) {
+    mode = buf.st_mode;
     if (S_ISCHR(mode) || S_ISFIFO(mode) || S_ISSOCK(mode)) {
       /*
       * XXX: is the following call interruptible? If so, this might
@@ -4559,11 +4863,11 @@
       }
     }
   }
-  if ((cur = ::lseek64(fd, 0L, SEEK_CUR)) == -1) {
+  if ((cur = ::lseek(fd, 0L, SEEK_CUR)) == -1) {
     return 0;
-  } else if ((end = ::lseek64(fd, 0L, SEEK_END)) == -1) {
+  } else if ((end = ::lseek(fd, 0L, SEEK_END)) == -1) {
     return 0;
-  } else if (::lseek64(fd, cur, SEEK_SET) == -1) {
+  } else if (::lseek(fd, cur, SEEK_SET) == -1) {
```

```
     return 0;
   }
   *bytes = end - cur;
@@ -4571,12 +4875,17 @@
 }

 int os::socket_available(int fd, jint *pbytes) {
-  // Linux doc says EINTR not returned, unlike Solaris
-  int ret = ::ioctl(fd, FIONREAD, pbytes);
+   if (fd < 0)
+     return OS_OK;
+
+   int ret;
+
+   RESTARTABLE(::ioctl(fd, FIONREAD, pbytes), ret);

-  //%% note ioctl can return 0 when successful, JVM_SocketAvailable
-  // is expected to return 0 on failure and 1 on success to the jdk.
-  return (ret < 0) ? 0 : 1;
+   //%% note ioctl can return 0 when successful, JVM_SocketAvailable
+   // is expected to return 0 on failure and 1 on success to the jdk.
+
+   return (ret == OS_ERR) ? 0 : 1;
 }

 // Map a block of memory.
@@ -4626,6 +4935,7 @@
   return munmap(addr, bytes) == 0;
 }

+#ifndef _ALLBSD_SOURCE
 static jlong slow_thread_cpu_time(Thread *thread, bool user_sys_cpu_time);

 static clockid_t thread_cpu_clockid(Thread* thread) {
@@ -4633,10 +4943,11 @@
   clockid_t clockid;

   // Get thread clockid
-  int rc = os::Linux::pthread_getcpuclockid(tid, &clockid);
+  int rc = os::Bsd::pthread_getcpuclockid(tid, &clockid);
   assert(rc == 0, "pthread_getcpuclockid is expected to return 0 code");
   return clockid;
 }
+#endif

 // current_thread_cpu_time(bool) and thread_cpu_time(Thread*, bool)
 // are used by JVM M&M and JVMTI to get user+sys or user CPU time
@@ -4646,39 +4957,71 @@
 // the fast estimate available on the platform.

 jlong os::current_thread_cpu_time() {
-  if (os::Linux::supports_fast_thread_cpu_time()) {
-    return os::Linux::fast_thread_cpu_time(CLOCK_THREAD_CPUTIME_ID);
+#ifdef __APPLE__
+  return os::thread_cpu_time(Thread::current(), true /* user + sys */);
+#elif !defined(_ALLBSD_SOURCE)
+  if (os::Bsd::supports_fast_thread_cpu_time()) {
+    return os::Bsd::fast_thread_cpu_time(CLOCK_THREAD_CPUTIME_ID);
   } else {
     // return user + sys since the cost is the same
     return slow_thread_cpu_time(Thread::current(), true /* user + sys */);
   }
+#endif
 }

 jlong os::thread_cpu_time(Thread* thread) {
+#ifndef _ALLBSD_SOURCE
   // consistent with what current_thread_cpu_time() returns
-  if (os::Linux::supports_fast_thread_cpu_time()) {
-    return os::Linux::fast_thread_cpu_time(thread_cpu_clockid(thread));
+  if (os::Bsd::supports_fast_thread_cpu_time()) {
```

```
+    return os::Bsd::fast_thread_cpu_time(thread_cpu_clockid(thread));
   } else {
     return slow_thread_cpu_time(thread, true /* user + sys */);
   }
+#endif
 }

 jlong os::current_thread_cpu_time(bool user_sys_cpu_time) {
-  if (user_sys_cpu_time && os::Linux::supports_fast_thread_cpu_time()) {
-    return os::Linux::fast_thread_cpu_time(CLOCK_THREAD_CPUTIME_ID);
+#ifdef __APPLE__
+  return os::thread_cpu_time(Thread::current(), user_sys_cpu_time);
+#elif !defined(_ALLBSD_SOURCE)
+  if (user_sys_cpu_time && os::Bsd::supports_fast_thread_cpu_time()) {
+    return os::Bsd::fast_thread_cpu_time(CLOCK_THREAD_CPUTIME_ID);
   } else {
     return slow_thread_cpu_time(Thread::current(), user_sys_cpu_time);
   }
+#endif
 }

 jlong os::thread_cpu_time(Thread *thread, bool user_sys_cpu_time) {
-  if (user_sys_cpu_time && os::Linux::supports_fast_thread_cpu_time()) {
-    return os::Linux::fast_thread_cpu_time(thread_cpu_clockid(thread));
+#ifdef __APPLE__
+  struct thread_basic_info tinfo;
+  mach_msg_type_number_t tcount = THREAD_INFO_MAX;
+  kern_return_t kr;
+  mach_port_t mach_thread;
+
+  mach_thread = pthread_mach_thread_np(thread->osthread()->thread_id());
+  kr = thread_info(mach_thread, THREAD_BASIC_INFO, (thread_info_t)&tinfo, &tcount);
+  if (kr != KERN_SUCCESS)
+    return -1;
+
+  if (user_sys_cpu_time) {
+    jlong nanos;
+    nanos = ((jlong) tinfo.system_time.seconds + tinfo.user_time.seconds) * (jlong)1000000000;
+    nanos += ((jlong) tinfo.system_time.microseconds + (jlong) tinfo.user_time.microseconds) * (jlong)1000;
+    return nanos;
+  } else {
+    return ((jlong)tinfo.user_time.seconds * 1000000000) + ((jlong)tinfo.user_time.microseconds * (jlong)1000);
+  }
+#elif !defined(_ALLBSD_SOURCE)
+  if (user_sys_cpu_time && os::Bsd::supports_fast_thread_cpu_time()) {
+    return os::Bsd::fast_thread_cpu_time(thread_cpu_clockid(thread));
   } else {
     return slow_thread_cpu_time(thread, user_sys_cpu_time);
   }
+#endif
 }

+#ifndef _ALLBSD_SOURCE
 //
 //  -1 on error.
 //
@@ -4702,7 +5045,7 @@
   FILE *fp;

   // We first try accessing /proc/<pid>/cpu since this is faster to
-  // process.  If this file is not present (linux kernels 2.5 and above)
+  // process.  If this file is not present (bsd kernels 2.5 and above)
   // then we open /proc/<pid>/stat.
   if ( proc_pid_cpu_avail ) {
     sprintf(proc_name, "/proc/%d/cpu", tid);
@@ -4722,12 +5065,12 @@
   }

   // The /proc/<tid>/stat aggregates per-process usage on
-  // new Linux kernels 2.6+ where NPTL is supported.
+  // new Bsd kernels 2.6+ where NPTL is supported.
```

```
   // The /proc/self/task/<tid>/stat still has the per-thread usage.
   // See bug 6328462.
   // There can be no directory /proc/self/task on kernels 2.4 with NPTL
   // and possibly in some other cases, so we check its availability.
-  if (proc_task_unchecked && os::Linux::is_NPTL()) {
+  if (proc_task_unchecked && os::Bsd::is_NPTL()) {
     // This is executed only once
     proc_task_unchecked = false;
     fp = fopen("/proc/self/task", "r");
@@ -4768,6 +5111,7 @@
     return (jlong)user_time * (1000000000 / clock_tics_per_sec);
   }
 }
+#endif

 void os::current_thread_cpu_time_info(jvmtiTimerInfo *info_ptr) {
   info_ptr->max_value = ALL_64_BITS;       // will not wrap in less than 64 bits
@@ -4784,11 +5128,17 @@
 }

 bool os::is_thread_cpu_time_supported() {
+#ifdef __APPLE__
   return true;
+#elif defined(_ALLBSD_SOURCE)
+  return false;
+#else
+  return true;
+#endif
 }

 // System loadavg support.  Returns -1 if load average cannot be obtained.
-// Linux doesn't yet have a (official) notion of processor sets,
+// Bsd doesn't yet have a (official) notion of processor sets,
 // so just return the system wide load average.
 int os::loadavg(double loadavg[], int nelem) {
   return ::getloadavg(loadavg, nelem);
@@ -4879,7 +5229,7 @@
 // abstime will be the absolute timeout time
 // TODO: replace compute_abstime() with unpackTime()

-static struct timespec* compute_abstime(timespec* abstime, jlong millis) {
+static struct timespec* compute_abstime(struct timespec* abstime, jlong millis) {
   if (millis < 0)  millis = 0;
   struct timeval now;
   int status = gettimeofday(&now, NULL);
@@ -4931,7 +5281,7 @@
         status = pthread_cond_wait(_cond, _mutex);
         // for some reason, under 2.7 lwp_cond_wait() may return ETIME ...
         // Treat this the same as if the wait was interrupted
-        if (status == ETIME) { status = EINTR; }
+        if (status == ETIMEDOUT) { status = EINTR; }
         assert_status(status == 0 || status == EINTR, status, "cond_wait");
       }
       -- _nParked ;
@@ -4983,16 +5333,16 @@
   // In that case, we should propagate the notify to another waiter.

   while (_Event < 0) {
-    status = os::Linux::safe_cond_timedwait(_cond, _mutex, &abst);
+    status = os::Bsd::safe_cond_timedwait(_cond, _mutex, &abst);
     if (status != 0 && WorkAroundNPTLTimedWaitHang) {
       pthread_cond_destroy (_cond);
       pthread_cond_init (_cond, NULL) ;
     }
     assert_status(status == 0 || status == EINTR ||
-                  status == ETIME || status == ETIMEDOUT,
+                  status == ETIMEDOUT,
                   status, "cond_timedwait");
     if (!FilterSpuriousWakeups) break ;                 // previous semantics
-    if (status == ETIME || status == ETIMEDOUT) break ;
+    if (status == ETIMEDOUT) break ;
```

```
     // We consume and ignore EINTR and spurious wakeups.
   }
   --_nParked ;
@@ -5051,7 +5401,7 @@
 // -----------------------------------------------------

 /*
- * The solaris and linux implementations of park/unpark are fairly
+ * The solaris and bsd implementations of park/unpark are fairly
   * conservative for now, but can be improved. They currently use a
   * mutex/condvar pair, plus a a count.
   * Park decrements count if > 0, else does a condvar wait.  Unpark
@@ -5066,7 +5416,7 @@
 #define NANOSECS_PER_MILLISEC 1000000
 #define MAX_SECS 100000000
 /*
- * This code is common to linux and solaris and will be moved to a
+ * This code is common to bsd and solaris and will be moved to a
   * common place in dolphin.
   *
   * The passed in time value is either a relative time in nanoseconds
@@ -5074,7 +5424,7 @@
   * into suitable seconds and nanoseconds components and stored in the
   * given timespec structure.
   * Given time is a 64-bit value and the time_t used in the timespec is only
- * a signed-32-bit value (except on 64-bit Linux) we have to watch for
+ * a signed-32-bit value (except on 64-bit Bsd) we have to watch for
   * overflow if times way in the future are given. Further on Solaris versions
   * prior to 10 there is a restriction (see cond_timedwait) that the specified
   * number of seconds, in abstime, is less than current_time  + 100,000,000.
@@ -5084,7 +5434,7 @@
   * years from "now".
   */

-static void unpackTime(timespec* absTime, bool isAbsolute, jlong time) {
+static void unpackTime(struct timespec* absTime, bool isAbsolute, jlong time) {
   assert (time > 0, "convertTime");

   struct timeval now;
@@ -5144,7 +5494,7 @@
   }

   // Next, demultiplex/decode time arguments
-  timespec absTime;
+  struct timespec absTime;
   if (time < 0 || (isAbsolute && time == 0) ) { // don't wait at all
     return;
   }
@@ -5180,7 +5530,7 @@
   // Don't catch signals while blocked; let the running threads have the signals.
   // (This allows a debugger to break into the running thread.)
   sigset_t oldsigs;
-  sigset_t* allowdebug_blocked = os::Linux::allowdebug_blocked_signals();
+  sigset_t* allowdebug_blocked = os::Bsd::allowdebug_blocked_signals();
   pthread_sigmask(SIG_BLOCK, allowdebug_blocked, &oldsigs);
 #endif

@@ -5191,14 +5541,14 @@
   if (time == 0) {
     status = pthread_cond_wait (_cond, _mutex) ;
   } else {
-    status = os::Linux::safe_cond_timedwait (_cond, _mutex, &absTime) ;
+    status = os::Bsd::safe_cond_timedwait (_cond, _mutex, &absTime) ;
     if (status != 0 && WorkAroundNPTLTimedWaitHang) {
       pthread_cond_destroy (_cond) ;
       pthread_cond_init    (_cond, NULL);
     }
   }
   assert_status(status == 0 || status == EINTR ||
-                status == ETIME || status == ETIMEDOUT,
+                status == ETIMEDOUT,
```

```
                status, "cond_timedwait");

 #ifdef ASSERT
@@ -5241,14 +5591,12 @@
 }


+/* Darwin has no "environ" in a dynamic library. */
+#ifdef __APPLE__
+#include <crt_externs.h>
+#define environ (*_NSGetEnviron())
+#else
 extern char** environ;
-
-#ifndef __NR_fork
-#define __NR_fork IA32_ONLY(2) IA64_ONLY(not defined) AMD64_ONLY(57)
-#endif
-
-#ifndef __NR_execve
-#define __NR_execve IA32_ONLY(11) IA64_ONLY(1033) AMD64_ONLY(59)
 #endif

 // Run the specified command in a separate process. Return its exit value,
@@ -5258,13 +5606,12 @@
 int os::fork_and_exec(char* cmd) {
   const char * argv[4] = {"sh", "-c", cmd, NULL};

-  // fork() in LinuxThreads/NPTL is not async-safe. It needs to run
+  // fork() in BsdThreads/NPTL is not async-safe. It needs to run
   // pthread_atfork handlers and reset pthread library. All we need is a
   // separate process to execve. Make a direct syscall to fork process.
   // On IA64 there's no fork syscall, we have to use fork() and hope for
   // the best...
-  pid_t pid = NOT_IA64(syscall(__NR_fork);)
-              IA64_ONLY(fork();)
+  pid_t pid = fork();

   if (pid < 0) {
     // fork failed
@@ -5273,15 +5620,14 @@
   } else if (pid == 0) {
     // child process

-    // execve() in LinuxThreads will call pthread_kill_other_threads_np()
+    // execve() in BsdThreads will call pthread_kill_other_threads_np()
     // first to kill every thread on the thread list. Because this list is
     // not reset by fork() (see notes above), execve() will instead kill
     // every thread in the parent process. We know this is the only thread
     // in the new process, so make a system call directly.
     // IA64 should use normal execve() from glibc to match the glibc fork()
     // above.
-    NOT_IA64(syscall(__NR_execve, "/bin/sh", argv, environ);)
-    IA64_ONLY(execve("/bin/sh", (char* const*)argv, environ);)
+    execve("/bin/sh", (char* const*)argv, environ);

     // execve failed
     _exit(-1);
--- src/os/linux/vm/os_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/os_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,15 +22,21 @@
  *
  */

-#ifndef OS_LINUX_VM_OS_LINUX_HPP
-#define OS_LINUX_VM_OS_LINUX_HPP
+#ifndef OS_BSD_VM_OS_BSD_HPP
+#define OS_BSD_VM_OS_BSD_HPP

-// Linux_OS defines the interface to Linux operating systems
+// Bsd_OS defines the interface to Bsd operating systems
```

```
-/* pthread_getattr_np comes with LinuxThreads-0.9-7 on RedHat 7.1 */
+/* pthread_getattr_np comes with BsdThreads-0.9-7 on RedHat 7.1 */
 typedef int (*pthread_getattr_func_type) (pthread_t, pthread_attr_t *);

-class Linux {
+#ifdef __APPLE__
+// Mac OS X doesn't support clock_gettime. Stub out the type, it is
+// unused
+typedef int clockid_t;
+#endif
+
+class Bsd {
   friend class os;

   // For signal-chaining
@@ -50,6 +56,7 @@
   static int sigflags[MAXSIGNUM];

   static int (*_clock_gettime)(clockid_t, struct timespec *);
+#ifndef _ALLBSD_SOURCE
   static int (*_pthread_getcpuclockid)(pthread_t, clockid_t *);

   static address   _initial_thread_stack_bottom;
@@ -61,6 +68,7 @@
   static bool _is_floating_stack;
   static bool _is_NPTL;
   static bool _supports_fast_thread_cpu_time;
+#endif

   static GrowableArray<int>* _cpu_to_node;

@@ -68,21 +76,27 @@

   static julong _physical_memory;
   static pthread_t _main_thread;
+#ifndef _ALLBSD_SOURCE
   static Mutex* _createThread_lock;
+#endif
   static int _page_size;

   static julong available_memory();
   static julong physical_memory() { return _physical_memory; }
   static void initialize_system_info();

+#ifndef _ALLBSD_SOURCE
   static void set_glibc_version(const char *s)      { _glibc_version = s; }
   static void set_libpthread_version(const char *s) { _libpthread_version = s; }
+#endif

   static bool supports_variable_stack_size();

+#ifndef _ALLBSD_SOURCE
   static void set_is_NPTL()                   { _is_NPTL = true;  }
-  static void set_is_LinuxThreads()           { _is_NPTL = false; }
+  static void set_is_BsdThreads()           { _is_NPTL = false; }
   static void set_is_floating_stack()         { _is_floating_stack = true; }
+#endif

   static void rebuild_cpu_to_node_map();
   static GrowableArray<int>* cpu_to_node()    { return _cpu_to_node; }
@@ -90,19 +104,27 @@
   static bool hugetlbfs_sanity_check(bool warn, size_t page_size);

  public:
+
   static void init_thread_fpu_state();
+#ifndef _ALLBSD_SOURCE
   static int  get_fpu_control_word();
   static void set_fpu_control_word(int fpu_control);
+#endif
   static pthread_t main_thread(void)                                { return _main_thread; }
```

```
+
+#ifndef _ALLBSD_SOURCE
   // returns kernel thread id (similar to LWP id on Solaris), which can be
   // used to access /proc
   static pid_t gettid();
   static void set_createThread_lock(Mutex* lk)                      { _createThread_lock = lk; }
   static Mutex* createThread_lock(void)                             { return _createThread_lock; }
+#endif
   static void hotspot_sigmask(Thread* thread);

+#ifndef _ALLBSD_SOURCE
   static address   initial_thread_stack_bottom(void)               { return _initial_thread_stack_bottom; }
   static uintptr_t initial_thread_stack_size(void)                 { return _initial_thread_stack_size; }
+#endif
   static bool is_initial_thread(void);

   static int page_size(void)                                       { return _page_size; }
@@ -114,14 +136,14 @@

   // For Analyzer Forte AsyncGetCallTrace profiling support:
   //
-  // This interface should be declared in os_linux_i486.hpp, but
+  // This interface should be declared in os_bsd_i486.hpp, but
   // that file provides extensions to the os class and not the
-  // Linux class.
+  // Bsd class.
   static ExtendedPC fetch_frame_from_ucontext(Thread* thread, ucontext_t* uc,
     intptr_t** ret_sp, intptr_t** ret_fp);

   // This boolean allows users to forward their own non-matching signals
-  // to JVM_handle_linux_signal, harmlessly.
+  // to JVM_handle_bsd_signal, harmlessly.
   static bool signal_handlers_are_installed;

   static int get_our_sigflags(int);
@@ -139,21 +161,23 @@
   static struct sigaction *get_chained_signal_action(int sig);
   static bool chained_handler(int sig, siginfo_t* siginfo, void* context);

+#ifndef _ALLBSD_SOURCE
   // GNU libc and libpthread version strings
   static const char *glibc_version()          { return _glibc_version; }
   static const char *libpthread_version()     { return _libpthread_version; }

-  // NPTL or LinuxThreads?
-  static bool is_LinuxThreads()               { return !_is_NPTL; }
+  // NPTL or BsdThreads?
+  static bool is_BsdThreads()              { return !_is_NPTL; }
   static bool is_NPTL()                       { return _is_NPTL;  }

-  // NPTL is always floating stack. LinuxThreads could be using floating
+  // NPTL is always floating stack. BsdThreads could be using floating
   // stack or fixed stack.
   static bool is_floating_stack()             { return _is_floating_stack; }

   static void libpthread_init();
   static bool libnuma_init();
   static void* libnuma_dlsym(void* handle, const char* name);
+#endif
   // Minimum stack size a thread can be created with (allowing
   // the VM to completely create the thread and enter user code)
   static size_t min_stack_allowed;
@@ -162,17 +186,21 @@
   static size_t default_stack_size(os::ThreadType thr_type);
   static size_t default_guard_size(os::ThreadType thr_type);

+#ifndef _ALLBSD_SOURCE
   static void capture_initial_stack(size_t max_size);

   // Stack overflow handling
   static bool manually_expand_stack(JavaThread * t, address addr);
```

```
   static int max_register_window_saves_before_flushing();
+#endif

   // Real-time clock functions
   static void clock_init(void);

+#ifndef _ALLBSD_SOURCE
   // fast POSIX clocks support
   static void fast_thread_clock_init(void);
+#endif

   static bool supports_monotonic_clock() {
     return _clock_gettime != NULL;
@@ -182,6 +210,7 @@
     return _clock_gettime ? _clock_gettime(clock_id, tp) : -1;
   }

+#ifndef _ALLBSD_SOURCE
   static int pthread_getcpuclockid(pthread_t tid, clockid_t *clock_id) {
     return _pthread_getcpuclockid ? _pthread_getcpuclockid(tid, clock_id) : -1;
   }
@@ -191,18 +220,19 @@
   }

   static jlong fast_thread_cpu_time(clockid_t clockid);
+#endif

   // Stack repair handling

   // none present

-  // LinuxThreads work-around for 6292965
+  // BsdThreads work-around for 6292965
   static int safe_cond_timedwait(pthread_cond_t *_cond, pthread_mutex_t *_mutex, const struct timespec
*_abstime);


-  // Linux suspend/resume support - this helper is a shadow of its former
+  // Bsd suspend/resume support - this helper is a shadow of its former
   // self now that low-level suspension is barely used, and old workarounds
-  // for LinuxThreads are no longer needed.
+  // for BsdThreads are no longer needed.
   class SuspendResume {
   private:
     volatile int _suspend_action;
@@ -335,4 +365,4 @@
     }
 } ;

-#endif // OS_LINUX_VM_OS_LINUX_HPP
+#endif // OS_BSD_VM_OS_BSD_HPP
--- src/os/linux/vm/os_linux.inline.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/os_bsd.inline.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,30 +22,30 @@
  *
  */

-#ifndef OS_LINUX_VM_OS_LINUX_INLINE_HPP
-#define OS_LINUX_VM_OS_LINUX_INLINE_HPP
+#ifndef OS_BSD_VM_OS_BSD_INLINE_HPP
+#define OS_BSD_VM_OS_BSD_INLINE_HPP

 #include "runtime/atomic.hpp"
 #include "runtime/os.hpp"
-#ifdef TARGET_OS_ARCH_linux_x86
-# include "atomic_linux_x86.inline.hpp"
-# include "orderAccess_linux_x86.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_x86
+# include "atomic_bsd_x86.inline.hpp"
+# include "orderAccess_bsd_x86.inline.hpp"
 #endif
```

```
-#ifdef TARGET_OS_ARCH_linux_sparc
-# include "atomic_linux_sparc.inline.hpp"
-# include "orderAccess_linux_sparc.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_sparc
+# include "atomic_bsd_sparc.inline.hpp"
+# include "orderAccess_bsd_sparc.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_zero
-# include "atomic_linux_zero.inline.hpp"
-# include "orderAccess_linux_zero.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_zero
+# include "atomic_bsd_zero.inline.hpp"
+# include "orderAccess_bsd_zero.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_arm
-# include "atomic_linux_arm.inline.hpp"
-# include "orderAccess_linux_arm.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_arm
+# include "atomic_bsd_arm.inline.hpp"
+# include "orderAccess_bsd_arm.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_ppc
-# include "atomic_linux_ppc.inline.hpp"
-# include "orderAccess_linux_ppc.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_ppc
+# include "atomic_bsd_ppc.inline.hpp"
+# include "orderAccess_bsd_ppc.inline.hpp"
 #endif

 // System includes
@@ -94,11 +94,19 @@

 inline bool os::allocate_stack_guard_pages() {
   assert(uses_stack_guard_pages(), "sanity check");
+#if !defined(__FreeBSD__) || __FreeBSD__ < 5
+  // Since FreeBSD 4 uses malloc() for allocating the thread stack
+  // there is no need to do anything extra to allocate the guard pages
+  return false;
+#else
+  // FreeBSD 5+ uses mmap MAP_STACK for allocating the thread stacks.
+  // Must 'allocate' them or guard pages are ignored.
   return true;
+#endif
 }


-// On Linux, reservations are made on a page by page basis, nothing to do.
+// On Bsd, reservations are made on a page by page basis, nothing to do.
 inline void os::split_reserved_memory(char *base, size_t size,
                                       size_t split, bool realloc) {
 }
@@ -126,7 +134,7 @@
 }

 inline jlong os::lseek(int fd, jlong offset, int whence) {
-  return (jlong) ::lseek64(fd, offset, whence);
+  return (jlong) ::lseek(fd, offset, whence);
 }

 inline int os::fsync(int fd) {
@@ -138,7 +146,7 @@
 }

 inline int os::ftruncate(int fd, jlong length) {
-  return ::ftruncate64(fd, length);
+  return ::ftruncate(fd, length);
 }

 inline struct dirent* os::readdir(DIR* dirp, dirent *dbuf)
@@ -147,7 +155,7 @@
   int status;
```

```
   assert(dirp != NULL, "just checking");

-  // NOTE: Linux readdir_r (on RH 6.2 and 7.2 at least) is NOT like the POSIX
+  // NOTE: Bsd readdir_r (on RH 6.2 and 7.2 at least) is NOT like the POSIX
   // version. Here is the doc for this function:
   // http://www.gnu.org/manual/glibc-2.2.3/html_node/libc_262.html

@@ -191,11 +199,11 @@
 }

 inline int os::close(int fd) {
-  return ::close(fd);
+  RESTARTABLE_RETURN_INT(::close(fd));
 }

 inline int os::socket_close(int fd) {
-  return ::close(fd);
+  RESTARTABLE_RETURN_INT(::close(fd));
 }

 inline int os::socket(int domain, int type, int protocol) {
@@ -231,7 +239,7 @@

     if (res == OS_ERR && errno == EINTR) {

-      // On Linux any value < 0 means "forever"
+      // On Bsd any value < 0 means "forever"

       if(timeout >= 0) {
         gettimeofday(&t, NULL);
@@ -255,14 +263,14 @@
 }

 inline int os::accept(int fd, struct sockaddr *him, int *len) {
-  // This cast is from int to unsigned int on linux.  Since we
+  // This cast is from int to unsigned int on bsd.  Since we
   // only pass the parameter "len" around the vm and don't try to
   // fetch it's value, this cast is safe for now. The java.net group
   // may need and want to change this interface someday if socklen_t goes
   // to 64 bits on some platform that we support.
-  // Linux doc says this can't return EINTR, unlike accept() on Solaris

-  return ::accept(fd, him, (socklen_t *)len);
+  // At least OpenBSD and FreeBSD can return EINTR from accept.
+  RESTARTABLE_RETURN_INT(::accept(fd, him, (socklen_t *)len));
 }

 inline int os::recvfrom(int fd, char *buf, int nBytes, int flags,
@@ -303,4 +311,4 @@
                             const char *optval, int optlen){
   return ::setsockopt(fd, level, optname, optval, optlen);
 }
-#endif // OS_LINUX_VM_OS_LINUX_INLINE_HPP
+#endif // OS_BSD_VM_OS_BSD_INLINE_HPP
--- src/os/linux/vm/os_share_linux.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/os_share_bsd.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,8 +22,8 @@
  *
  */

-#ifndef OS_LINUX_VM_OS_SHARE_LINUX_HPP
-#define OS_LINUX_VM_OS_SHARE_LINUX_HPP
+#ifndef OS_BSD_VM_OS_SHARE_BSD_HPP
+#define OS_BSD_VM_OS_SHARE_BSD_HPP

 // misc
 void signalHandler(int, siginfo_t*, ucontext_t*);
@@ -34,4 +34,4 @@

 #define PROCFILE_LENGTH 128
```

```
-#endif // OS_LINUX_VM_OS_SHARE_LINUX_HPP
+#endif // OS_BSD_VM_OS_SHARE_BSD_HPP
--- src/os/linux/vm/perfMemory_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/perfMemory_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -27,7 +27,7 @@
 #include "memory/allocation.inline.hpp"
 #include "memory/resourceArea.hpp"
 #include "oops/oop.inline.hpp"
-#include "os_linux.inline.hpp"
+#include "os_bsd.inline.hpp"
 #include "runtime/handles.inline.hpp"
 #include "runtime/perfMemory.hpp"
 #include "utilities/exceptions.hpp"
@@ -132,14 +132,14 @@

 // Shared Memory Implementation Details

-// Note: the solaris and linux shared memory implementation uses the mmap
+// Note: the solaris and bsd shared memory implementation uses the mmap
 // interface with a backing store file to implement named shared memory.
 // Using the file system as the name space for shared memory allows a
 // common name space to be supported across a variety of platforms. It
 // also provides a name space that Java applications can deal with through
 // simple file apis.
 //
-// The solaris and linux implementations store the backing store file in
+// The solaris and bsd implementations store the backing store file in
 // a user specific temporary directory located in the /tmp file system,
 // which is always a local file system and is sometimes a RAM based file
 // system.
@@ -690,7 +690,7 @@
 // memory region on success or NULL on failure. A return value of
 // NULL will ultimately disable the shared memory feature.
 //
-// On Solaris and Linux, the name space for shared memory objects
+// On Solaris and Bsd, the name space for shared memory objects
 // is the file system name space.
 //
 // A monitoring application attaching to a JVM does not need to know
Files src/os/linux/vm/stubRoutines_linux.cpp and src/os/bsd/vm/stubRoutines_bsd.cpp are identical
--- src/os/linux/vm/threadCritical_linux.cpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/threadCritical_bsd.cpp        2011-07-26 20:21:21.000000000 -0600
@@ -24,7 +24,7 @@

 #include "precompiled.hpp"
 #include "runtime/threadCritical.hpp"
-#include "thread_linux.inline.hpp"
+#include "thread_bsd.inline.hpp"

 // put OS-includes here
 # include <pthread.h>
--- src/os/linux/vm/thread_linux.inline.hpp        2011-07-26 20:21:21.000000000 -0600
+++ src/os/bsd/vm/thread_bsd.inline.hpp        2011-07-26 20:21:21.000000000 -0600
@@ -22,41 +22,41 @@
  *
  */

-#ifndef OS_LINUX_VM_THREAD_LINUX_INLINE_HPP
-#define OS_LINUX_VM_THREAD_LINUX_INLINE_HPP
+#ifndef OS_BSD_VM_THREAD_BSD_INLINE_HPP
+#define OS_BSD_VM_THREAD_BSD_INLINE_HPP

 #include "runtime/atomic.hpp"
 #include "runtime/prefetch.hpp"
 #include "runtime/thread.hpp"
 #include "runtime/threadLocalStorage.hpp"
-#ifdef TARGET_OS_ARCH_linux_x86
-# include "atomic_linux_x86.inline.hpp"
-# include "orderAccess_linux_x86.inline.hpp"
-# include "prefetch_linux_x86.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_x86
```

```
+# include "atomic_bsd_x86.inline.hpp"
+# include "orderAccess_bsd_x86.inline.hpp"
+# include "prefetch_bsd_x86.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_sparc
-# include "atomic_linux_sparc.inline.hpp"
-# include "orderAccess_linux_sparc.inline.hpp"
-# include "prefetch_linux_sparc.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_sparc
+# include "atomic_bsd_sparc.inline.hpp"
+# include "orderAccess_bsd_sparc.inline.hpp"
+# include "prefetch_bsd_sparc.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_zero
-# include "atomic_linux_zero.inline.hpp"
-# include "orderAccess_linux_zero.inline.hpp"
-# include "prefetch_linux_zero.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_zero
+# include "atomic_bsd_zero.inline.hpp"
+# include "orderAccess_bsd_zero.inline.hpp"
+# include "prefetch_bsd_zero.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_arm
-# include "atomic_linux_arm.inline.hpp"
-# include "orderAccess_linux_arm.inline.hpp"
-# include "prefetch_linux_arm.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_arm
+# include "atomic_bsd_arm.inline.hpp"
+# include "orderAccess_bsd_arm.inline.hpp"
+# include "prefetch_bsd_arm.inline.hpp"
 #endif
-#ifdef TARGET_OS_ARCH_linux_ppc
-# include "atomic_linux_ppc.inline.hpp"
-# include "orderAccess_linux_ppc.inline.hpp"
-# include "prefetch_linux_ppc.inline.hpp"
+#ifdef TARGET_OS_ARCH_bsd_ppc
+# include "atomic_bsd_ppc.inline.hpp"
+# include "orderAccess_bsd_ppc.inline.hpp"
+# include "prefetch_bsd_ppc.inline.hpp"
 #endif

 // Contains inlined functions for class Thread and ThreadLocalStorage

 inline void ThreadLocalStorage::pd_invalidate_all() {} // nothing to do

-#endif // OS_LINUX_VM_THREAD_LINUX_INLINE_HPP
+#endif // OS_BSD_VM_THREAD_BSD_INLINE_HPP
Files src/os/linux/vm/vmError_linux.cpp and src/os/bsd/vm/vmError_bsd.cpp are identical
```